UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Sustitución sensorial con motores de vibración

Máster Universitario en Ingeniería de Telecomunicación

Autor: Torre Albarsanz, José Antonio
Tutor: González de Rivera Peces, Guillermo

FECHA: Septiembre, 2018

SUSTITUCIÓN SENSORIAL CON MOTORES DE VIBRACIÓN

AUTOR: José Antonio Torre Albarsanz
TUTOR: Guillermo González de Rivera Peces

Hardware and Control Technology Laboratory

HCTLab

Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Septiembre de 2018

Resumen

En el Trabajo de Fin de Master que se presenta a continuación, se ha diseñado, construido y programado un prototipo para un Dispositivo de Sustitución Sensorial, del cual, investigadores del Laboratorio de Visión y Percepción de la Facultad de Psicología de la UAM, ya habían realizado un prototipo inicial, muy costoso, lento de procesamiento y sin optimizar.

Un Dispositivo de Sustitución Sensorial son dispositivos que generalmente están formados por sensores encargados de recopilar información del entorno e interpretarla y traducirla en estimulaciones de uno de los sentidos humanos. El prototipo diseñado en este TFM se trata de un Dispositivo de Sustitución Sensorial encargado de traducir la información visual del entorno mediante cámaras, para posteriormente traducirlo a diferentes vibraciones mediante una matriz de motores colocados en un chaleco vestible.

Siguiendo los requisitos marcados por el equipo de investigación antes mencionado, y basándose en el prototipo ya existente, se ha realizado un nuevo diseño optimizado, donde se ha mantenido el uso de Xbee como método de comunicación inalámbrica entre la unidad de procesamiento de imágenes y el prototipo.

Para la comunicación y control de cada uno de los motores que forman el chaleco se continúa usando el protocolo de comunicación I2C, a través del cual se mandan las direcciones y comandos de cada uno de los dispositivos esclavos que activan y controlan los motores de forma independiente.

Además de la fabricación del prototipo, se ha realizado el software de control que lleva cargado el microprocesador. El cometido de este software es realizar la interpretación de los datos enviados por la unidad de procesamiento de imágenes y traducirlos a vibraciones de diferentes intensidades en cada uno de los motores, consiguiendo así una transformación del entorno visual a un entorno táctil.

Palabras Clave

Dispositivo de Sustitución Sensorial, Prototipo, Actuadores vibrotáctiles, I2C, TWI, USART, ATMEGA32U4, XBEE, PCA9685.

Abstract

In the Project presented below, a prototype for a Sensorial Substitution Dispositive has been designed, built and programmed. There was a preliminary version developed by the researchers from the Vision and Perception Laboratory of the UAM's School of Psychology that was very expensive, slow processing and not optimized.

A Sensorial Substitution Device is a device that is generally formed by sensors responsible for collecting information from the environment and interpreting and translating it into stimulations of one of the human senses. The prototype designed in this TFM is a Sensorial Substitution Device in charge of translating the visual information of the environment through cameras, to later translate it to different vibrations by means of a matrix of motors placed in a wearable vest.

Following the requirements set by this research team, and based on the existing prototype, a new optimized design has been carried out, in which the use of Xbee as a method of wireless communication between the image processing unit and the prototype has been implemented.

For the communication and control of each of the motors on the vest, the I2C communication protocol is implemented, through which are sent the addresses and commands of each of the slave devices that activate and control the motors independently.

In addition to the manufacture of the prototype, the control software carried by the microprocessor has been made. The main purpose of this software is to perform the interpretation of the data sent by the image processing unit and translate it into vibrations of different intensities in each of the motors, thus achieving a transformation of the visual environment to a tactile environment.

Key Words

Sensory Substitution Device, Prototype, Vibrotactile Actuators, I2C, TWI, USART, ATMEGA32U4, XBEE, PCA9685.

Agradecimientos

Gracias a todos los que me habéis ayudado y apoyado durante esta etapa de mis estudios. Especialmente quiero agradecérselo a mi tutor, Guillermo González de Rivera, por su ayuda y su paciencia. También quiero agradecérselo a mis amigos Álvaro y Raúl, que han estado conmigo durante todo el master, y por último agradecer a mi familia y a Claudia la paciencia que han tenido conmigo aguantándome cada día.

Muchas gracias.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción.

En 1960 comienza a surgir y popularizarse los *dispositivos de sustitución sensorial* o SSD, los cuales son unos dispositivos con los que se pretende ayudar o bien sustituir por completo un sentido defectuoso, como puede ser la vista, por otro de los sentidos, como puede ser el tacto o el oído, de forma que la información que recibiría el sentido defectuoso se "*traduzca*" en información comprensible por el otro sentido.

Los SSD son usados en un amplio rango de situaciones. Los más popularizados son los SSD vibrotáctiles, los cuales pueden ser útiles en situaciones donde no se dispone de visibilidad, o poca visibilidad, como en el caso de los bomberos en sitios con mucho humo, pilotos en condiciones meteorológicas adversas, o personas ciegas simplemente.

En 2010, el número estimado de personas con dificultad visual fue de 285 millones, de los cuales se estima que 39 millones se corresponden a personas ciegas. Comparado con el número de posibles usuarios, el número de SSD que hay disponibles en el mercado y/o que en la actualidad están siendo usados es bajo.

Debido a la gran aplicabilidad que tienen en la actualidad este tipo de dispositivos, el equipo de investigadores del Laboratorio de Visión y Percepción de la Facultad de Psicología de la UAM conjuntamente con el *Hardware and Control Technology Laboratory* (HCTLab), quieren mejorar el entendimiento, futuro y aplicabilidad de los dispositivos SSD mediante el desarrollo e investigación de uno nuevo, en el cual se centra este Trabajo de Fin de Master.

1.1 Motivación

Este TFM parte de un proyecto de colaboración con investigadores del Laboratorio de Visión y Percepción de la Facultad de Psicología de la UAM. En este laboratorio se ha desarrollado un equipo de sustitución sensorial, cuyo objetivo se centra en poder transmitir a una persona ciega el lugar y distancia de los objetos que le rodean a través de una matriz de motores vibradores colocados en el pecho del sujeto.

El prototipo desarrollado actualmente, es un prototipo muy limitado, pesado de llevar, muy poco manejable y con un gran consumo de energía.

Por ello el objetivo de este TFM consiste en realizar un desarrollo utilizando la tecnología actual, mucho más ligero, con bajo consumo, un diseño modular, fácil de actualizar y con comunicación inalámbrica, permitiendo dotarle de muchas más prestaciones.

1.2 Objetivos

El objetivo principal de este TFM es, partiendo del desarrollo actual, mejorar todas las características de éste, reduciendo considerablemente las dimensiones del prototipo, así como su consumo de energía. Para alcanzar este objetivo principal, se proponen los siguientes objetivos parciales:

- Leer y entender cómo funciona el prototipo desarrollado por los investigadores del Laboratorio de Visión y Percepción de la Facultad de Psicología de la UAM.

- Búsqueda y elección de los dispositivos electrónicos que formarán parte del nuevo prototipo, teniendo en cuenta que deben ser dispositivos que mejoren las características del prototipo actual.
- Poner en funcionamiento los diferentes dispositivos electrónicos elegidos desde una plataforma conocida para posteriormente conectarlos a un microprocesador de mayor potencia.
- Realizar un diseño modular del prototipo, compuesto por dos partes diferenciables, una primera placa formada por el microprocesador y el dispositivo de comunicación inalámbrica (ZigBee), y una segunda placa formada por un driver I2C encargado de activar los diferentes motores de la matriz.
- Poner en funcionamiento el nuevo prototipo elaborado.
- Evaluar el funcionamiento y consumo del prototipo una vez se encuentre en funcionamiento.

1.3 Organización de la memoria

La documentación de este TFM, se estructura de la siguiente manera:

El capítulo primero hace una pequeña introducción sobre el tema de los dispositivos de sustitución sensorial. Además, se describen los motivos por los cuales surge este TFM, así como los objetivos que se persiguen.

Seguidamente, el capítulo segundo repasa el estado del arte de los dispositivos de sustitución sensorial, el cual se centra en los sistemas táctiles, más concretamente en los táctiles-visuales. A continuación, se describen varios prototipos creados para la sustitución sensorial, siendo alguno de ellos similares al prototipo en el que se centra este TFM, para finalizar con varios prototipos comerciales que se pueden encontrar en el mercado y su precio.

En el capítulo tercero, se da una breve introducción al prototipo del cual parte este TFM y que es lo que se quiere conseguir. Para ello, se describen cuáles son las características principales que se necesitan en cada uno de los componentes que formaran el prototipo a desarrollar, dando varias alternativas de cada dispositivo y explicando el motivo de los dispositivos seleccionados, dando una visión de las características más importantes y útiles de cada uno, así como de los protocolos de comunicación que se van a usar para el desarrollo.

En el capítulo cuarto, se describen cada una de las etapas que se han llevado a cabo para el desarrollo de un primer prototipo, partiendo de unas primeras etapas de desarrollo hardware en las cuales se han diseñado los PCBs que contendrán y conectarán los dispositivos seleccionados, y otras etapas de desarrollo de librerías software necesarias para el control de los diferentes módulos y buses disponibles en el microprocesador elegido. Finalmente, en este cuarto capítulo, se ha analizado el código fuente principal creado para el control del prototipo, además de realizar un diseño de un PCB para el driver I2C mucho más optimizado e industrializable.

Para finalizar, en el capítulo quinto, se presentan las conclusiones alcanzadas y se proponen posibles trabajos futuros, así como ampliaciones del prototipo diseñado.

2 Estado del arte

2.1 Historia

La idea de sustitución sensorial surgió en los años 60 por Paul Bach-y-Rita [2] como un método de usar una modalidad sensorial, principalmente el tacto, para obtener información ambiental y ser utilizada por otra modalidad sensorial, principalmente la vista.

El primer sistema de sustitución sensorial fue desarrollado por Bach-y-Rita como un medio de plasticidad cerebral en individuos con ceguera congénita.

Después de esta invención histórica, la sustitución sensorial ha sido la base de muchos estudios que investigan la neurociencia cognitiva y perceptiva. Desde entonces, la sustitución sensorial ha contribuido al estudio de la función cerebral, la cognición humana y la rehabilitación.

2.2 Sistemas de Sustitución Sensorial

En los sistemas de sustitución sensorial, generalmente se tienen sensores que recopilan datos del entorno. Estos datos se transmiten a un sistema de acoplamiento que interpreta y transduce la información para luego traducirla en un estimulador. Este estimulador finalmente estimula una modalidad sensorial funcional.

Dentro de los sistemas de sustitución sensorial, pueden encontrarse tres tipos de sistemas principalmente, los cuales son:

- Sistemas Táctiles.
- Sistemas Auditivos.
- Implantes de Sistemas Nerviosos.

Este TFM se centra en un tipo de Sistema Táctil, por lo que únicamente se dará una pequeña visión de los sistemas táctiles y cómo funcionan, obviando los otros dos tipos de sistemas.

2.2.1 Sistemas Táctiles.

Para comprender la sustitución sensorial táctil, es esencial comprender alguna fisiología básica de los receptores táctiles de la piel. Por ello hay cinco tipos básicos de receptores táctiles:

- Corpúsculo de Pacinia.
- Corpúsculo de Meissner.
- Terminaciones de Ruffini.
- Terminaciones nerviosas de Merkel.
- Terminaciones nerviosas libres.

Estos receptores se caracterizan principalmente por el tipo de estímulo que mejor los activa y por su velocidad de adaptación a los estímulos sostenidos.

Por ello existen dos tipos diferentes de estimuladores, los estimuladores electrotáctiles y los vibrotáctiles.

Los estimuladores electrotáctiles usan estimulación eléctrica directa de la terminación nerviosa en la piel para iniciar los potenciales de acción. La sensación desencadenada, la quemadura, el picor, el dolor, la presión, etc. dependen del voltaje estimulante.

Con los sistemas de estimulación electrotáctil, muchos factores afectan la sensación desencadenada: voltaje estimulante, corriente, forma de onda, tamaño del electrodo, material, fuerza de contacto, ubicación de la piel, grosor e hidratación. La estimulación electrotactiva puede implicar la estimulación directa de los nervios (percutánea) o a través de la piel (transcutánea).

Los estimuladores electrostáticos se exploran como dispositivos de interacción humano-computadora para pantallas táctiles. Estos se basan en un fenómeno llamado electrovibración, que permite que las corrientes a nivel de microamperios se sientan como rugosidades en una superficie.

Figura 2-1: Estimulador electrotáctil.

Por otro lado, los estimuladores vibrotáctiles usan presión y las propiedades de los mecanorreceptores de la piel para iniciar potenciales de acción.

Los sistemas vibrotáctiles usan las propiedades de los mecanorreceptores en la piel, por lo que tienen menos parámetros que necesitan ser monitoreados en comparación con la estimulación electrotáctil. Sin embargo, los sistemas de estimulación vibrotáctil deben tener en cuenta la rápida adaptación del sentido táctil.

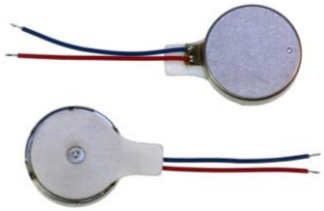


Figura 2-2: Estimulador Vibrotáctil.

A continuación, se describen varios sistemas actuales de sustitución táctil:

- **Táctil-Visual:**

Una de las primeras y más conocidas formas de dispositivos de sustitución sensorial fue el TVSS de Paul Bach-y-Rita [2][3] que convirtió la imagen de una cámara de video en una imagen táctil y la acopló a los receptores táctiles en la parte posterior de un sujeto ciego.

Recientemente, se han desarrollado varios sistemas nuevos que conectan la imagen táctil con receptores táctiles en diferentes áreas del cuerpo, como el pecho, la frente, la yema del dedo, el abdomen y la frente. La imagen táctil es producida por cientos de activadores colocados en la persona. En experimentos, sujetos ciegos (o vendados) equipados con TVSS pueden aprender a detectar formas y orientarse.

- **Táctil-Auditivo.**
- **Táctil-Vestibular.**
- **Táctil-táctil para restaurar la sensación periférica.**

En este TFM se han desarrollado un dispositivo de sustitución táctil-visual, que irá colocado, a forma de chaleco vestible, en el pecho de un sujeto ciego, con el cual se conseguirá orientar y evitar obstáculos de su entorno. Por ello, se va a proporcionar una breve visión de los dispositivos que existen a día de hoy, enfocados al mismo objetivo.

2.3 Prototipos SSD Táctil-Visual.

Tal y como se ha dicho anteriormente, uno de los primeros y más conocidos dispositivos de sustitución sensorial fue el desarrollado en 1969 por Paul Bach-y-Rita [2][3], abriendo así un campo de investigación para muchos, donde se han propuesto diversas soluciones para conseguir un mismo objetivo. A continuación, se va a describir brevemente cada solución para posteriormente poder situar nuestro prototipo entre todos los ya desarrollados.

2.3.1 Sustitución de la Visión por Proyección de una Imagen Táctil – Bach-y-Rita (TVSS).

El dispositivo creado por Bach-y-Rita [2][3] está formado por 400 estimuladores solenoides dispuestos en una matriz de 20x20, los cuales se colocaron en una silla típica de dentista.

Los estimuladores estaban colocados con una separación de 12mm. Además, los estimuladores tenían una punta de Teflón de 1mm, los cuales se colocaban vibrando sobre la piel de la espalda del sujeto (Figura 2-3).

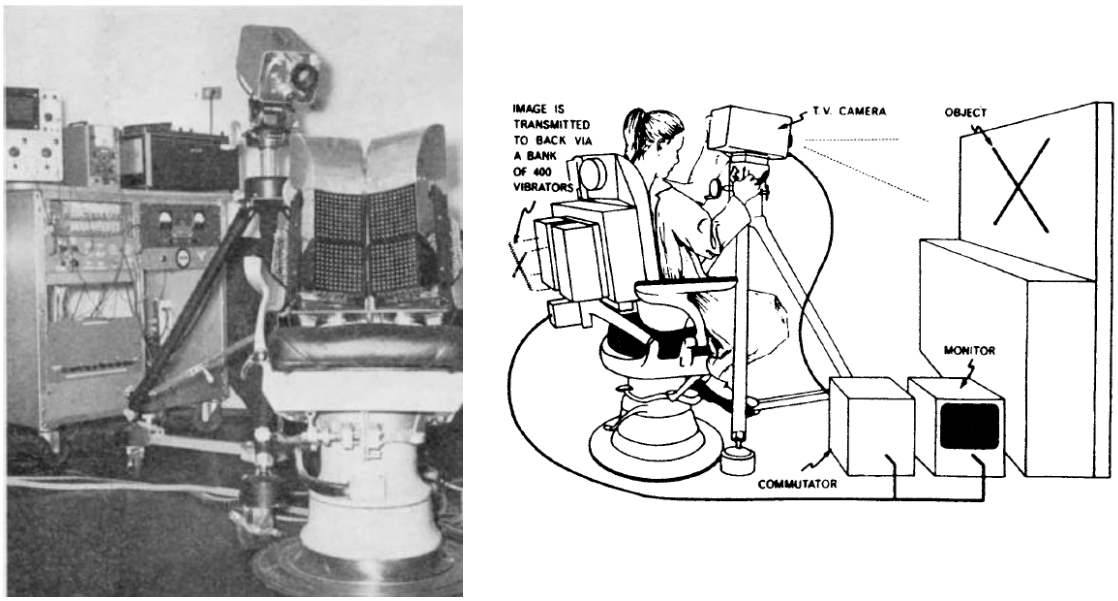


Figura 2-3: Dispositivo desarrollado por Bach-y-Rita.

La actividad de estos dispositivos, es decir si estaban encendidos o apagados, podía ser monitorizada visualmente en un osciloscopio que +traba un dibujo en dos dimensiones sobre su pantalla.

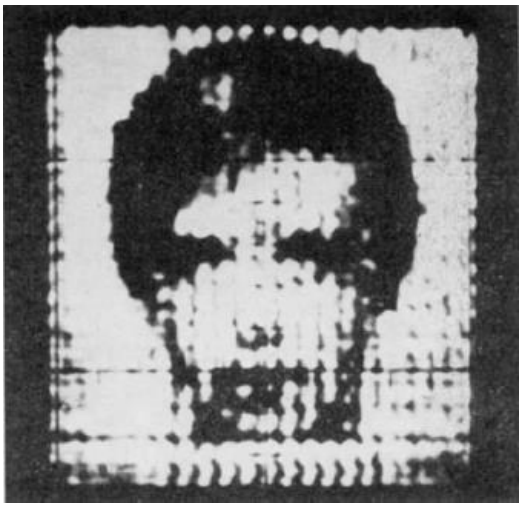


Figura 2-4: Osciloscopio con imagen 2D mostrando la actividad de encendido o apagado de los estimuladores.

El sujeto manipulaba una cámara de televisión montada sobre un trípode, el cual escaneaba el objeto colocado en una mesa enfrente de él. El sujeto podía apuntar la cámara, equipada con una lente de zoom, a las diferentes partes de la sala, localizando o identificando objetos o personas.

Una vez construido el dispositivo, se realizaron varias pruebas con 6 sujetos ciegos, y tras varias horas de entrenamiento, se obtuvieron los siguientes resultados en cuanto a precisión por parte de los sujetos averiguando el objeto frente a la cámara y el tiempo que necesitaban para averiguarlo.

Tabla 1. Precisión y Latencia en las respuestas de sujetos ciegos y con vista normal.

Precisión

	Orientación de Línea	**Inclinación del tablero**
Ciegos N=6	99.6%	82.9% $\sigma = 10.4$
De vista normal N=6	100%	97.5% $\sigma = 1.5$
Significante	No significante	$P < 0.01$

Latencia en Segundos

	Orientación de Línea	**Inclinación del tablero**
Ciegos N=6	1.2	8.4 $\sigma = 2.9$
De vista normal N=6	1.1	2.8 $\sigma = 0.7$
Significado	No significante	$P < 0.001$

2.3.2 VideoTact.

El *VideoTact* I[4] es una tecnología de matriz electrotáctil multicanal que consiste en un sustrato laminado *Kapton*/*Kapton* conductor delgado, flexible y relativamente económico, con electrodos de titanio.

El *VideoTact* está compuesto por 16 canales de generación de forma de onda simultáneos, cada uno de los cuales puede abordarse independientemente. Cada canal maneja hasta 48 electrodos multiplexados y tiene 768 electrodos en total en una matriz de 32 columnas x 24 filas (Figura 2-5). Es capaz de interactuar con una placa de captura de imágenes basada en PC para el procesamiento de imágenes de la cámara de video y tiene un subsistema basado en un procesador de señal digital. El sistema está controlado por una interfaz basada en MS-Windows y tiene un analizador de lenguaje de scripts para su uso en experimentos.

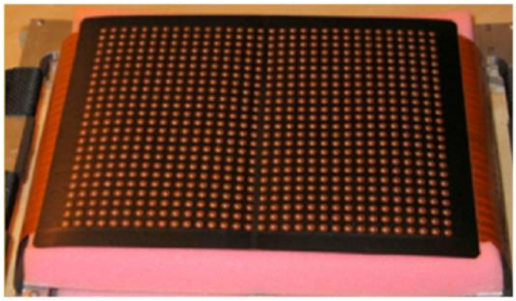


Figura 2-5: Lamina con electrodos de Titanio del dispositivo VideoTact.

2.3.3 TSAS (Tactile Situation Awareness System).

Los estímulos vibrotáctiles proporcionados por TSAS [5] pueden proporcionar señales inminentes, es decir, indicaciones de que un objeto se acerca al usuario incluso cuando el usuario no puede ver el objeto (por ejemplo, debido a distracción durante las operaciones del vehículo o pérdida sensorial asociada con la ceguera).

En segundo lugar, TSAS se ha adaptado para proporcionar sutiles señales vibrotáctiles a pacientes con trastornos del equilibrio y personas mayores con insensibilidad a las señales posturales vestibulares endógenas. El sistema detecta cuando una persona con dicha pérdida sensorial entra en un estado de inestabilidad postural y proporciona retroalimentación vibrotáctil para que una persona pueda tomar acción correctiva antes de que ocurra una caída.

En tercer lugar, TSAS se está adaptando para proporcionar señales de navegación para pilotos en entornos ruidosos y para pilotos con pérdida de audición inducida por ruido.

Este sistema está formado por 22 vibradores neumáticos vibrando a 50Hz, o vibradores electromecánicos vibrando a 150Hz, situados en un cinturón elástico, el cual va conectado a una unidad de control alimentado por una batería recargable (Figura 2-6).

Figura 2-6: Array de vibradores del sistema TSAS.

2.3.4 Personal Tactile Navigator.

El *Personal Tactile Navigator (PTN)* [6] consiste en una pantalla táctil, un paquete de sensores, una unidad de procesamiento y un paquete de baterías. La pantalla táctil consta de ocho elementos vibratorios (motores vibrantes de CC de 1.3 V, alojados en cajas de PVC rectangular) con un área de contacto de 1.5 por 2.0 cm y una frecuencia de vibración de 155 Hz. Las cajas están montadas en una pretina ajustable (Figura 2-7). La resolución de las pantallas (es decir, ocho actuadores táctiles para 360 grados) se encuentra entre el mínimo requerido para la navegación (es decir, dos elementos: uno para la izquierda y otro para la derecha) y el límite de percepción de la dirección en el torso. Las ubicaciones de los elementos en el cinturón también son ajustables, por lo que pueden colocarse fácilmente en la dirección de los ejes cardinales y oblicuos independientemente de la forma corporal del participante. El cinturón se usa sobre la ropa interior.

Figura 2-7: Sistema *Personal Tactile Navigator* (PTN).

La unidad de procesamiento, el sensor GPS, la brújula electrónica y la pantalla táctil reciben su energía de una batería NiMH recargable (14.4 V, 3.3 Ah, fusionada). Para garantizar la recepción adecuada del GPS, el sensor GPS se usa en la parte superior del torso. La velocidad de actualización de la unidad GPS es de 1 Hz; sin embargo, el sistema funcionaba a 10 Hz o

más. Esto significa que los datos de la brújula digital podrían usarse para señalar una desviación del rumbo incluso sin una actualización GPS en la ubicación exacta. Esto permite a la PTN detectar cambios en el rumbo sin un cambio de ubicación (por ejemplo, al dar la vuelta al eje) y proporcionar retroalimentación durante los cambios rápidos del rumbo.

El sistema se usaba como un sistema de GPS o brújula para seguir una ruta hasta un punto determinado del mapa. De esta forma el cinturón con los elementos vibratorios generaba vibraciones dependiendo de la dirección a seguir.

2.3.5 CyberTouch

El guante CyberTouch desarrollado por Immersion Corporation [7] para uso en aplicaciones de entorno virtual tiene seis actuadores vibrotáctiles programables individualmente, uno montado en la parte posterior de cada dedo y otro en la palma. La configuración de la mano se registra desde 22 sensores que están ubicados sobre las articulaciones de los dedos y la muñeca, y las mediciones de estos sensores se utilizan para controlar la posición y configuración de una mano virtual presentada al usuario en una pantalla visual. Los actuadores vibrotáctiles conectados al guante se activan siempre que la mano virtual interactúa con un objeto virtual, que proporciona información sobre qué dedos están en contacto con el objeto.

Figura 2-8: Guante CyberTouch

En la actualidad, estas pantallas táctiles entregan entradas vibrotáctiles bastante simples a la mano en frecuencias únicas que se encuentran dentro del rango de sensibilidades máximas.

2.3.6 NavBelt.

Navbelt fue desarrollado en 1992 por Borenstein y la Universidad de Michigan [8] como un sistema de guía, que utiliza un sistema de evitación de obstáculos de robot móvil. El prototipo, está compuesto por sensores de rango ultrasónicos, un ordenador y unos auriculares. El ordenador recibe información de los ocho sensores ultrasónicos (Figura 2-9) y crea un mapa de los ángulos (cada uno para cada sensor) y la distancia de cualquier objeto en este ángulo. Luego, el algoritmo de evitación de obstáculos produce sonidos apropiados para cada modo.

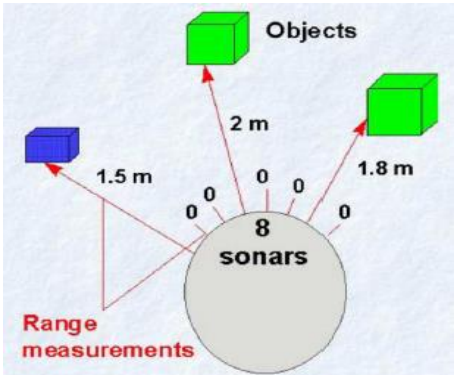


Figura 2-9: Esquema de funcionamiento de NavBelt.

Navbelt tiene dos modos: el modo de guía y el modo de imagen.

Durante el modo de guía, la computadora conoce el destino del usuario y con un solo pitido recurrente lo guía en la dirección óptima de viaje generada. Pero en la práctica, una implementación realista (no simulación) requeriría más sensores.

En el modo de imagen, se tocan ocho tonos de diferentes amplitudes en rápida sucesión desde ocho direcciones virtuales diferentes (similar a un barrido de radar). La computadora traduce (según el modo) estos mapas a sonidos que el usuario puede escuchar desde sus auriculares. Las desventajas de los sistemas son el uso de retroalimentación de audio (exclusivamente), el prototipo voluminoso y que a los usuarios se requieren largos períodos de entrenamiento.

2.3.7 Proyecto de la Universidad de Stuttgart.

Investigadores de la Universidad de Stuttgart en Alemania [9] desarrollaron un sistema portátil que ayuda a las personas ciegas a orientarse en ambientes interiores. El prototipo consiste en un módulo sensor con un bastón desmontable y un ordenador portátil. El sensor (Figura 2-10) está equipado con dos cámaras, un teclado (similar a los teléfonos móviles), una brújula digital, un detector de inclinación 3D y un altavoz. Se puede manejar como una linterna y "Al presionar las teclas designadas, se pueden elegir diferentes opciones de secuencia y sonoridad y las consultas sobre las características de un objeto se pueden enviar al ordenador portátil. Después de una evaluación exitosa, estas consultas se responden acústicamente sobre un motor de texto a voz".

Figura 2-10: Prototipo de la Universidad de Stuttgart.

El ordenador contiene el software necesario para la detección de la distancia de detección de color y el tamaño de los objetos y las capacidades de la red inalámbrica de área local (WLAN). El dispositivo funciona casi en tiempo real. Para mejorar el rendimiento del sistema, se creó un modelo 3-D virtual del entorno, por lo que la información del sensor se puede combinar con los datos almacenados en el modelo 3-D.

En conclusión, los aspectos positivos del sistema son la robustez del sensor, el funcionamiento casi en tiempo real y la amabilidad del usuario. Lo negativo es que la operación de retención y escaneo y, hasta el momento, las pruebas simuladas son limitadas.

2.3.8 Asistencia de navegación para personas con discapacidad visual.

Sainarayanan de la Universidad de Malasia Sabah [10] desarrolló un prototipo (NAVI) para ayudar a las personas ciegas a identificar los obstáculos durante la navegación, identificando los objetos que están frente a ellos. El prototipo de asistencia de navegación para discapacitados visuales (Figura 2-11) consiste en una cámara de video digital, casco (sostiene la cámara), auriculares estéreo, el sistema de procesamiento, baterías recargables y un chaleco (que contiene procesador y baterías).

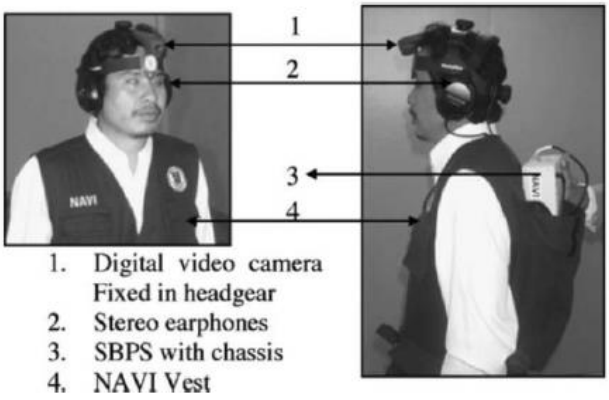


Figura 2-11: Prototipo NAVI y sus componentes.

La idea es que los humanos se concentren en los objetos que se encuentran frente al centro de visión, por lo que es importante distinguir entre el fondo y los obstáculos. La cámara de video captura video en escala de grises, que se remuestrea a una resolución de 32 × 32. Luego, utilizando una red neuronal de cuantificación de vector de aprendizaje difusa, los píxeles se clasifican en fondo u objetos utilizando diferentes características de nivel de gris. Luego, los píxeles del objeto se mejoran y el fondo se suprime. La etapa final corta la imagen procesada en partes izquierda y derecha, se transforma en sonido (estéreo) que se envía al usuario a través de los auriculares.

La ventaja de este sistema es que el prototipo está desarrollado y es operativo y en tiempo real. Las desventajas son el uso de comentarios de audio y que no se proporciona información sobre las distancias de los objetos.

2.3.9 Proyecto de la Universidad de Guelph.

Zelek y estudiantes de la Universidad de Guelph [11], en Canadá, desarrolló un dispositivo económico, construido con componentes listos para usar, portátiles y de baja potencia que

transformarán la información de profundidad (salida de cámaras estéreo) en información táctil o auditiva para su uso. por personas con discapacidad visual durante la navegación.

El prototipo (Figura 2-12) se compone de dos cámaras estéreo, una unidad táctil (guante con cinco zumbadores piezoeléctricos en cada punta del dedo) y un ordenador portátil. Cada dedo corresponde a una dirección espacial. Por ejemplo, el dedo medio corresponde al frente recto. Usando un algoritmo de estereovisión estándar, se crea el mapa de profundidad y luego se divide en cinco secciones verticales, cada una correspondiente a un elemento de vibración. Si un píxel en un área corresponde a una distancia umbral, entonces se activa el elemento de vibración correspondiente, informando al usuario sobre un obstáculo cercano en esa dirección.

Figura 2-12: Prototipo de la Universidad de Guelph.

La baja potencia/coste son las ventajas, pero la falta de metodologías sofisticadas (por ejemplo, el algoritmo de estereovisión necesita mejoras) no ofrece resultados interesantes.

2.3.10 Sistema de visión de electrón-neuronal.

El sistema de visión electrón-neuronal (ENVS) de Meers and Ward de la Universidad de Wollongong en Australia [12] tiene como objetivo lograr la evitación de obstáculos y la navegación en entornos al aire libre con la ayuda de sensores visuales, GPS y simulación electrotáctil. El prototipo (Figura 2-13) consiste en un auricular con dos cámaras estéreo y brújula digital, un procesador portátil con capacidades de GPS y una base de datos de puntos de referencia, la unidad de estimulación nerviosa eléctrica transcutánea (TENS) (microcontrolador) y los guantes TENS.

Figura 2-13: ENVS y sus componentes.

El concepto básico detrás del prototipo ENVS son las cámaras estéreo, utilizando visión estereoscópica, crear un mapa de profundidad del entorno y utilizando el procesador portátil, la información sobre los obstáculos (del mapa de profundidad) o puntos de referencia (del

GPS) se transforma mediante TENS para pulsos eléctricos que estimulan los nervios en la piel a través de electrodos ubicados en los guantes de datos TENS. La cantidad de estimulación es directamente proporcional a la distancia de los objetos en la dirección apuntada por cada dedo.

El prototipo se probó con usuarios con los ojos vendados en un entorno de campus al aire libre, trabajando en tiempo real (video de 15 cuadros/s). Con un entrenamiento mínimo (1 h), los usuarios pudieron informar la ubicación de los obstáculos, evitarlos y llegar a un destino predefinido El sistema es uno de los más completos aquí descritos porque es portátil, en tiempo real, tiene capacidades de GPS, no bloquea la audición del usuario, y los primeros resultados experimentales son muy prometedores. Algunos de los inconvenientes son que no se detectan objetos en el suelo o que sobresalgan, que se requiere una trayectoria plana (es decir, sin escaleras ni pendientes) y que el usuario debe usar los guantes TENS.

2.3.11 Sistema de Visión Táctil.

El objetivo de Johnson y Higgins de la Universidad de Arizona [13] era crear un dispositivo portátil que convirtiera la información visual en señal táctil para ayudar a las personas con discapacidad visual a navegar por sí mismas evitando obstáculos.

El prototipo se llama sistema de visión táctil (TVS) (Figura 2-14) y consiste en un cinturón con 14 motores vibradores espaciados lateralmente, un cinturón de cámara con dos cámaras web y un ordenador portátil en una mochila.

Figura 2-14: Prototipo TVS.

Se crea un mapa de profundidad 2-D usando las imágenes de las dos cámaras. Luego se divide en 14 regiones verticales. A cada motor vibrador se le asigna una región y el valor del objeto más cercano en cada región se transforma en vibración (Figura 2-15). La frecuencia de vibración y la distancia del objeto no son lineales (aumenta drásticamente para los objetos más cercanos) y los objetos muy lejanos o muy cercanos se ignoran. La información dada por el cinturón de tacto se aplica sobre la piel del abdomen (plana, grande, de fácil acceso, sin interferencia con otras funciones de navegación del usuario). El video se captura con una velocidad de hasta 10 cuadros/s, lo que hace que el sistema sea en tiempo real para las velocidades de marcha normales.

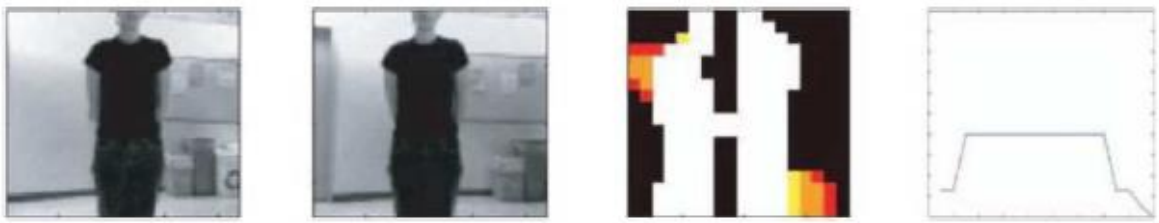


Figura 2-15: Modo de operación del prototipo TVS.

Las principales ventajas de TVS son que es usable, le da al usuario manos libres sin bloquear la audición. y funciona en tiempo real. Las desventajas son que no se puede diferenciar entre obstáculos que sobresalen y de tierra y que no se han realizado experimentos reales con personas con discapacidades visuales.

2.3.12 Proyecto EPFL.

Cardin de la Escuela Politécnica Federal de Lausanne (EPFL) [14] desarrolló un sistema portátil que detecta los obstáculos en la altura del hombro a través de un sistema de sonar estereoscópico y envía pulsos vibrotáctiles para informar al usuario sobre su localización.

El prototipo consiste en sensores de sonda, un microcontrolador, ocho vibradores y una consola de calibración (PDA).

El microcontrolador (Figura 2-16) recopila información de los sonares (Figura 2-17) proporcional a la distancia del obstáculo detectado. Calcula la distancia aproximada del obstáculo y luego convierte la distancia a una señal de modulación por ancho de pulso (PWM) que se redirige a los vibradores (diferentes velocidades de vibración), de modo que el usuario puede ser informado para la detección.

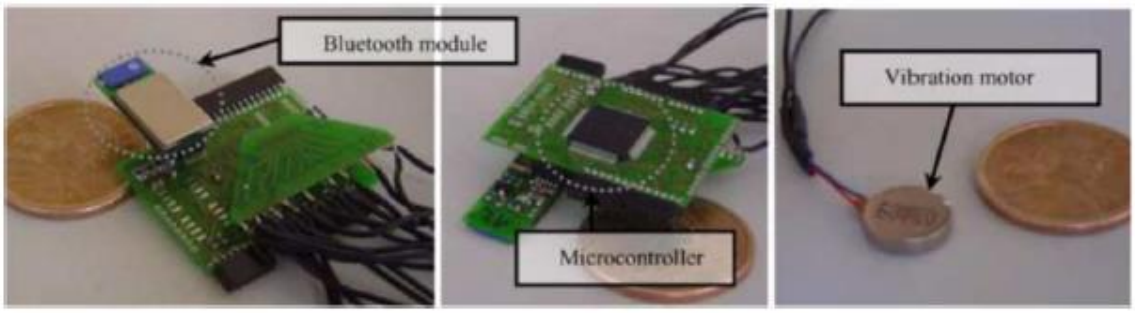


Figura 2-16: Hardware del prototipo EPFL.

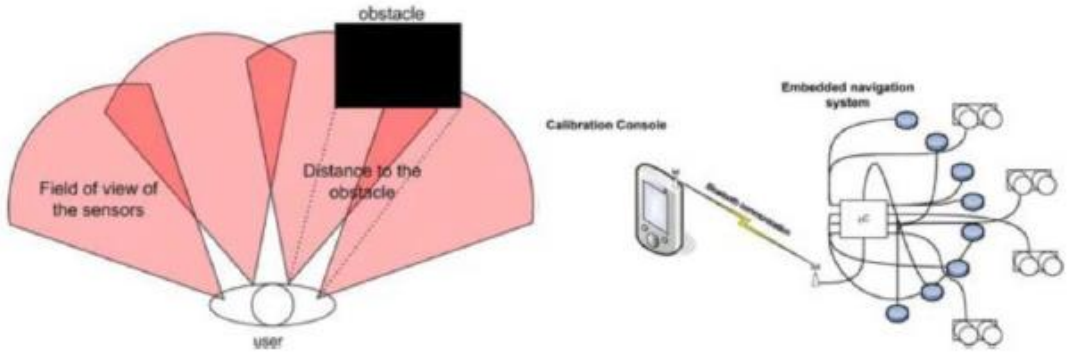


Figura 2-17: Operación y diseño a alto nivel del prototipo EPFL.

Los sonares y los vibradores están montados en la ropa del usuario, comenzando por un hombro y terminando por el otro. Finalmente, la consola de calibración se comunica con el microcontrolador a través de Bluetooth y permite la modificación dinámica de la curva de calibración (distancia real entre el objeto y el sensor).

Los resultados experimentales se obtuvieron probando el dispositivo en un ambiente interior controlado (corredor con personas que caminan y puertas que se abren y cierran) en 5 usuarios. Los resultados fueron alentadores ya que los usuarios lograron después de una pequeña capacitación caminar por el pasillo, distinguir los obstáculos (que están en el lado izquierdo o en el lado derecho) y ubicarse en el pasillo.

Las ventajas de este proyecto son que es un sistema de luz portátil, bajo consumo de energía y de bajo costo. Las desventajas son que no se prueba en personas con discapacidad visual y que cuatro sonares no pueden representar adecuadamente el espacio 3-D (diferentes alturas). Otro problema práctico mencionado por los autores es la interferencia de las manos y su detección como obstáculos.

2.3.13 Tyflos.

El sistema de navegación Tyflos fue concebido por Bourbakis a mediados de la década de 1990 y se han desarrollado varios prototipos [15][16][17]. El sistema de navegación Tyflos está formado por dos módulos básicos: el Reader y el Navigator.

El objetivo principal del sistema Tyflos es integrar diferentes tecnologías de ayuda a la navegación, como un ordenador de mano inalámbrico, cámaras, sensores de rango, sensores de GPS, micrófonos, procesadores de lenguaje natural, dispositivos de texto a voz y un grabador de audio digital, etc. y metodologías tales como la segmentación basada en regiones, la conversión de datos de rango, la fusión, etc., para ofrecer a los invidentes más independencia durante la navegación y la lectura.

Los dispositivos de entrada audiovisuales y los dispositivos de salida de audio y táctil pueden ser usados (o transportados) por el usuario. Los datos recopilados por los sensores son procesados por los módulos de Tyflos, cada uno especializado en una o más tareas. En particular, interactúa con sensores externos (como GPS, sensores de rango, etc.) y con el usuario, facilitando la entrega de contenido personalizado y enfocado. El usuario comunica la tarea de interés al asistente de movilidad, utilizando un esquema de interacción multimodal. La función principal del navegador es capturar los datos ambientales de varios

sensores y asignar el contenido extraído y procesado a las interfaces de usuario disponibles de la manera más adecuada.

El último prototipo de sistema de navegador Tyflos desarrollado en la Universidad Estatal de Wright (Figura 2-18). Consta de dos cámaras, un altavoz, un micrófono, un chaleco de vibración 2-D (conectado en el abdomen del usuario) controlado por un microprocesador y un ordenador portátil, e integra varios componentes de software y hardware [16][17].

Figura 2-18: Segundo prototipo del Sistema Tyflos.

Las cámaras estéreo crean un mapa de profundidad del entorno (que puede verificarse mediante la salida del sensor de rango). Un algoritmo de resolución alta reduce la resolución del mapa de profundidad a baja resolución, manteniendo la información necesaria para la navegación, como rutas de navegación seguras y objetos de interés (objetos en movimiento y personas, usando detección de movimiento y metodologías de detección de rostros) [17]. Esta "imagen" final es una representación del espacio 3-D, y se convierte en detección de vibración en una matriz/chaleco de vibración 2-D que se adjunta en el abdomen o el pecho del usuario. El elemento de la matriz que vibra representa la dirección, donde se detecta un objeto y los diferentes niveles de vibración representan la distancia del objeto [16] (Figura 2-19). La retroalimentación de audio opcional puede informar al usuario sobre objetos de interés.

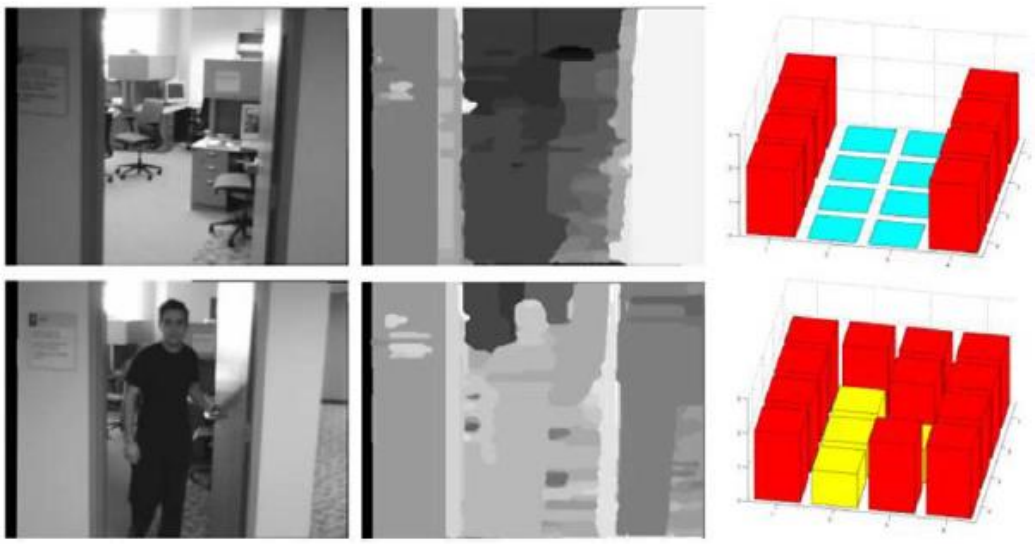


Figura 2-19: Operación de los Tyflos con dos escenarios de navegación.

Las principales ventajas de los Tyflos son las orejas libres y que el uso del conjunto de vibración 2-D con las frecuencias de vibración variable ofrece al usuario una representación más precisa del entorno 3-D (incluidos los obstáculos de altura del suelo y la cabeza) también información para distancias. Las desventajas son que el sistema aún no se ha probado en usuarios ciegos, lo cual es un paso importante para recibir comentarios para futuros cambios de hardware y software.

2.3.14 Productos Comerciales.

Hay varios productos comerciales disponibles en el mercado. Sus funcionalidades son limitadas y tienen un pequeño valor científico y tecnológico. Además, su costo es relativamente alto y no son ampliamente aceptados por los usuarios. Por lo tanto, se van a presentar algunos de ellos con una pequeña descripción, sin entrar en un análisis más profundo.

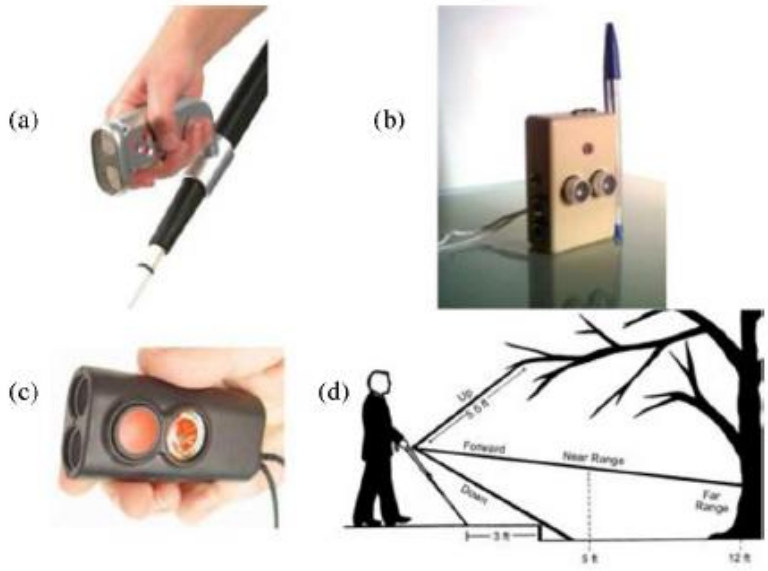


Figura 2-20: Productos Comerciales: a) K-Sonar Cane; b) Mini-radar; c) Mini-guide; d) LaserCane.

K-Sonar Cane [20] es un dispositivo que se adjunta en bastones blancos tradicionales (Figura 2-20 (a)). Consiste en un sensor de rango ultrasónico (sonar) y un microprocesador que convierte las distancias al sonido que el usuario puede escuchar a través de los auriculares. Los objetos distantes están relacionados con sonidos de tono alto y objetos cercanos a sonidos de bajo tono. Su precio es de aproximadamente $700.

Mini-Radar [21] es un dispositivo (Figura 2-20 (b)) que usa el sonar para detectar obstáculos frontales. Produce mensajes de audio en el idioma cuando se detecta un objeto. También puede proporcionar información sobre la distancia del objeto. Otra función es la "estabilidad direccional" que ayuda al usuario a caminar recto sin cambiar su dirección. Tiene un precio de aproximadamente $600.

Miniguide [22] es un dispositivo pequeño como una luz de mano (Figura 2-20 (c)) que indica la distancia al objeto más cercano, a través de su tasa de vibración. Tiene múltiples modos y rangos (hasta 8 m). Cuanto más rápida es la velocidad de vibración, más cerca está el objeto.

La ayuda tiene una toma de auriculares que puede proporcionar información de audio. Tiene un precio de aproximadamente $330.

LaserCane [23] es un bastón con tres sensores de rango láser: para la altura de la cabeza, la recta y los obstáculos de caída (Figura 2-20 (d)), y un sistema de audio que produce sonidos de advertencia (o que corresponden a la distancia de obstáculos) y estimuladores de vibración para advertencias. El usuario puede seleccionar entre sonido, vibración o ambos. Tiene un precio de aproximadamente $3000.

Ultracane [24] también es un bastón con escáneres integrados de alcance láser. Si se detecta un obstáculo, ciertos botones de vibración advierten al usuario. Hay diferentes vibraciones para diferentes direcciones y diferentes velocidades de vibración dependiendo de la distancia del obstáculo. Su precio es de aproximadamente $900.

3 Diseño

Los investigadores del Laboratorio de Visión y Percepción de la Facultad de Psicología de la UAM se centraron en dispositivos con actuadores vibrotáctiles sobre la piel. Como se ha descrito anteriormente, existen muchos dispositivos que funcionan de esta forma, como el primer prototipo creado por Bach-y-Rita de TVSS [2][3], el cual activaba una densa serie de actuadores conectados a una silla. El TVSS acopló información del entorno (intensidad de luz en 400 regiones de un campo de cámara) con la activación de los actuadores de una matriz de 20x20 de actuadores táctiles, permitiendo a los usuarios el control basado en joystick de la cámara.

El prototipo que se ha creado inicialmente, está basado en el TVSS de Bach-y-Rita, pero con varias diferencias, ya que la estimulación de los actuadores vibrotáctiles de Bach-y-Rita, se realizaba en función de la intensidad de luz capturada por la cámara en 400 regiones, correspondiendo cada región a un actuador. En cambio, el prototipo creado, se basa en la distancia del obstáculo para activar los actuadores. A continuación, se va a describir el prototipo creado, desde el hardware que lo forma hasta el funcionamiento de este.

3.1 Prototipo inicial.

El dispositivo de sustitución sensorial desarrollado, al cual se llamó "*Tactile-Sight"* (TSIGHT) [25] tiene como objetivo tener un alto número de actuadores vibrotáctiles para presentar el flujo a través de variaciones en el área de estimulación y la intensidad de la vibración.

El dispositivo tiene la siguiente estructura (Figura 3-1):

- Un sistema de adquisición del entorno mediante cámaras (Kinect).
- Unidad de procesamiento, realizada por un ordenador.
- Comunicación inalámbrica mediante el estándar de comunicación ZigBee entre la unidad de procesamiento y la unidad de control.
- Un microprocesador que actúa como dispositivo maestro en la comunicación I2C.
- Varios microprocesadores que actúan como esclavos que comunican con el maestro mediante I2C y accionan 6 actuadores cada uno.
- Actuadores vibrotáctiles.

El dispositivo Kinect proporciona información en tiempo real al escenario central sobre la distancia a la superficie más cercana para cada región a la vista, capturando la distancia de los objetos en el entorno.

Por otro lado, los módulos de potencia y control son modulares, los cuales están interconectados mediante una red I2C, donde existe un dispositivo maestro, el cual recibe la información, de forma inalámbrica, sobre la intensidad de activación de cada actuador y reenvía dicha información al módulo esclavo correspondiente.

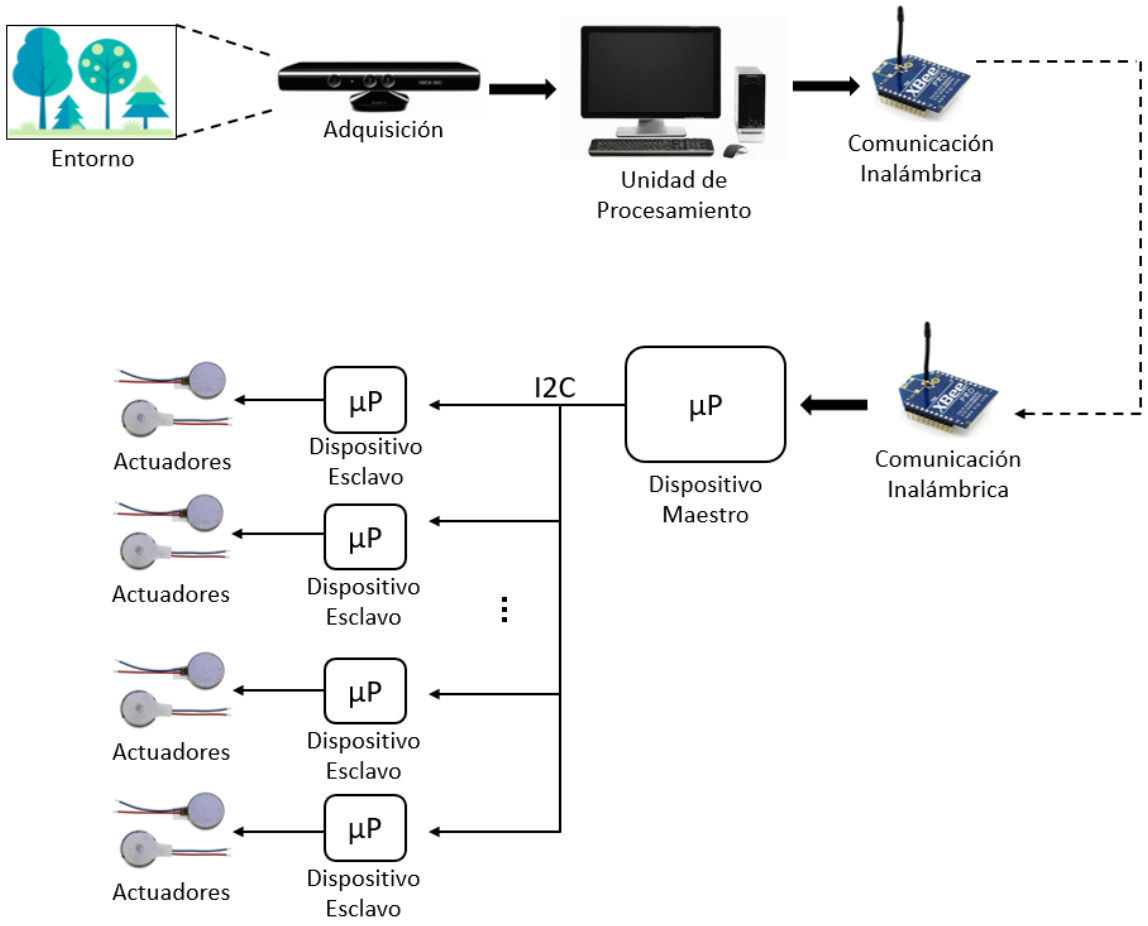


Figura 3-1: Arquitectura dispositivo TSIGHT.

Como resultado, se obtuvo un prototipo funcional, pero con unas dimensiones poco manejables (Figura 3-2). Además, el diseño, si se aumentaba considerablemente el número de actuadores, también aumentaría el tiempo de proceso y de transmisión en el bus I2C, lo que podría resultar incómodo para su uso.

Por este motivo, surgió la necesidad de desarrollar un nuevo prototipo que redujese considerablemente la cantidad de cables para interconectar el prototipo y poder hacer un dispositivo con mayor portabilidad.

A continuación, se va a describir el diseño que ha sido creado en este TFM, detallando cada uno de los componentes que lo forman.

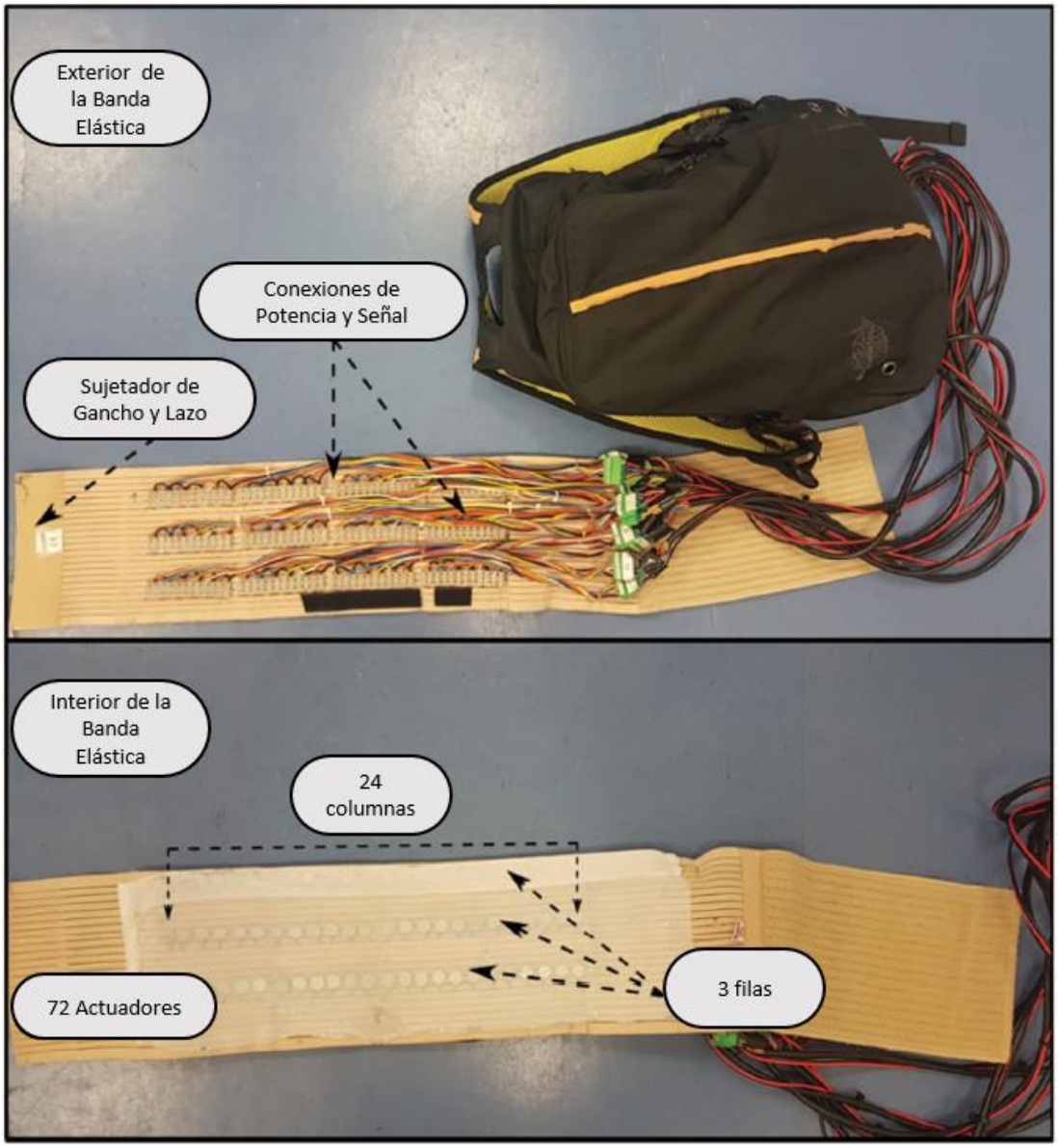


Figura 3-2: Prototipo del Laboratorio de Visión y Percepción de la Facultad de Psicología de la UAM.

3.2 Prototipo desarrollado.

Para este prototipo, se siguió el mismo esquema que en el prototipo inicial (Figura 3-1), el cual únicamente se centra en la parte del dispositivo que forma el chaleco, dejando tal y como está la parte de adquisición y procesamiento de las imágenes.

Es por ello, que una parte importante de este TFM es la elección de los dispositivos que van a formar el chaleco, los cuales tienen que proporcionar un factor de mejora al prototipo ya existente. Para ello, a continuación, se van a describir las características principales que deben tener los componentes que formaran el chaleco:

- **Dispositivo de Comunicación Inalámbrica:** Un elemento muy importante en el diseño es el dispositivo de comunicación inalámbrica que llevará a cabo la comunicación entre el ordenador donde se procesan las imágenes y el microprocesador que llevará incorporado el chaleco. Para ello se seguirá manteniendo la comunicación a través de un dispositivo de la familia Zigbee.

- **Microprocesador:** El microprocesador, que funcionará como dispositivo maestro, es importantísimo que implemente un módulo UART, el cual es el encargado de realizar una comunicación serie entre el dispositivo Xbee y el propio microprocesador, y el bus I2C, a través del cual se activaran cada uno de los motores vibradores. Además, es muy importante que disponga de una velocidad de operación que no ralentice el funcionamiento del conjunto, y con un encapsulado sencillo y fácil de conectar, con un tamaño reducido.

- **Driver I2C:** Este dispositivo será el dispositivo esclavo en la comunicación I2C, y a él, irán directamente conectados los motores de vibración que formarán el chaleco. Este dispositivo debe implementar, como es obvio, el bus I2C, el cual permita conectar al mismo bus varios drivers, de manera que esto no limite la cantidad de motores que pueden formar el chaleco. Además, debe tener varias salidas para poder controlar con un solo driver varios dispositivos vibradores. Dichas salidas deben tener la capacidad de generar señales PWM (Pulse Width Modulation) para poder alimentar a los vibradores de forma que vibren a diferentes intensidades de vibración, así como una intensidad de salida suficiente para poder hacer funcionar los motores de vibración.

3.2.1 Búsqueda de componentes.

En este apartado se describen los diferentes chips disponibles en el mercado que encajan en mayor o menor medida con las especificaciones básicas buscadas para el diseño del lector de tarjetas. Algunos se desechan por no disponer de alguna de las funcionalidades requeridas para el diseño y otros, aunque sus características son suficientes, se desechan por no encontrarse disponibles en los suministradores habituales o porque su precio está fuera de lugar en comparación con chips similares.

A continuación, se muestra un resumen de los chips estudiados junto con un extracto de sus principales características.

3.2.1.1 Dispositivo de comunicación inalámbrica.

Para la comunicación inalámbrica se valoraron diversas posibilidades, entre las cuales destacan:

- Del fabricante *Digi International* se tienen toda la familia de dispositivos Xbee [26], los cuales se pueden encontrar de diversos modelos (con antena integrada, sin antena, a diferentes frecuencias de funcionamiento, diferentes consumos de potencia, variedad en los rangos de alcance, etc.). Todos ellos (o prácticamente todos) usan el estándar de comunicación ZigBee, y tienen el mismo encapsulado, además de tener implementado un módulo UART, para la comunicación serie con el microprocesador. Como ventajas en estos dispositivos se encuentra la variedad de dispositivos, usando todos el mismo encapsulado, por lo que, en el futuro, podría sustituirse por modelos más potentes, si fuese necesario, o por modelos nuevos (siempre y cuando mantengan el mismo tipo de encapsulado), si el modelo usado quedase obsoleto. Por otra parte, tienen una plataforma de configuración propia, *XCTU*, la cual facilita considerablemente su configuración y la comunicación entre ellos. Por último, estos dispositivos tienen un precio entre los 20 euros, el más sencillo de la familia, hasta unos 35 euros.

- Del fabricante *Silicon Labs*, se analizaron unos módulos Zigbee con un microprocesador Cortex ARM integrado, el cual dispone de interfaz I2C y UART, el dispositivo MGM12P [27]. La gran ventaja que tienen estos módulos, es que se eliminaría la necesidad de tener un microprocesador aparte, reduciendo considerablemente el tamaño del prototipo a desarrollar. A pesar de tener un microprocesador integrado, estos módulos no tienen un precio elevado, ya que rondan entre los 15€-30€. Los inconvenientes de estos módulos son que tienen un montaje bastante complicado, y por lo que, con las herramientas disponibles en el laboratorio no se podría soldar en ningún PCB, y además son bastante nuevos y no se dispone de documentación suficiente.

3.2.1.2 Microprocesador.

Para el microprocesador se analizaron dos plataformas de desarrollo de pequeño tamaño, que son las siguientes:

- Las plataformas de desarrollo Teensy, de las cuales se pueden encontrar varias versiones con diferentes microprocesadores, y todas ellas con interfaz I2C y UART, se analizó únicamente la versión Teensy 2.0, la cual está compuesta por un microprocesador ATMEGA32U4 de 8 bits a 16MHz [28], que son características más que suficientes para el prototipo que se va a desarrollar. Además, esta versión tiene un precio aproximado de 16€ y unas dimensiones bastante pequeñas. Por último, al ser un microprocesador de la familia Atmel, se dispone de software propietario bastante conocido, Atmle Studio, en la que se puede programar el micro tanto en lenguaje C como lenguaje Arduino.
- Del fabricante BeagleBoard, se disponía de una plataforma de desarrollo BeagleBone Pocket [29], la cual trae integrado un microprocesador Cortex ARM A-8 a 1GHz, 512 MHz de RAM, dos interfaces I2C y 3 UART. Esta placa tiene un precio aproximado de 25€ y aunque también tiene unas dimensiones bastante pequeñas, son más grandes que la plataforma anterior.

3.2.1.3 Driver I2C.

Para el control de los motores mediante un bus I2C, se encontraron las siguientes posibilidades:

- Del fabricante NXP, se analizó el chip PCA9685, el cual consiste en un driver I2C, que da la posibilidad de controlar 16 leds, o en este caso motores, donde las salidas pueden configurarse como PWM. Además, se pueden conectar 64 drivers a un mismo bus I2C, por lo que se podrían controlar los 128 motores necesarios con 8 drivers. Por último, estos drivers entregan una corriente en cada salida de 25mA, lo cual es suficiente para controlar los motores.

- Del fabricante Texas Instruments, se analizó el chip DRV8830 [31]. Este chip dispone de 2 salidas, por lo que únicamente se podrían conectar 2 motores a él, y en el mismo bus I2C se pueden conectar 9 chips. Por lo tanto, no sería posible controlar los 128 motores necesarios. En este caso, se proporciona una corriente de salida de 1A, que está muy por encima de lo necesario para los motores que se van a utilizar.

3.2.2 Selección de Componentes.

En esta sección se describen las principales características de los componentes seleccionados para formar parte del prototipo que se está diseñando.

3.2.2.1 Dispositivos de comunicación inalámbrica.

Como dispositivo de comunicación inalámbrica se ha decidido utilizar los dispositivos de la familia Xbee de *Digi International* [26], ya que proporcionan las características buscadas a bajo precio. Además, estos dispositivos son altamente conocidos y utilizados en este tipo de aplicaciones.

Estos módulos utilizan el protocolo de red llamado IEEE 802.15.4 para crear redes FAST POINT-TO-MULTIPOINT (punto a multipunto) o para redes PEER-TO-PEER (punto a punto). Están pensados para comunicaciones con un alto tráfico de datos con una baja latencia, algo muy importante en este caso, ya que es imprescindible obtener los estímulos de forma casi inmediata.

El modelo seleccionado en concreto es el modelo Xbee S1 Pro con *Whip Antenna* (la antena es un pequeño cable que sobresale) (Figura 3-3), aunque podría cambiarse por cualquier otro modelo con el mismo encapsulado.

Figura 3-3: Xbee S1 Pro con Whip Antenna.

Este dispositivo dispone de un puerto Serie con el que se enviarán los datos al microprocesador, además de 8 entradas y salidas digitales combinadas con otras posibilidades como entradas/salidas analógicas, así como señales de control (RST, CST y Reset).

Las antenas de este modelo en concreto funcionan a 2.4GHz, algo que no tiene por qué ser así en otros modelos. Es por eso y por otros factores que el fabricante no recomienda mezclar estos dispositivos utilizando diferentes modelos en cada punto de la comunicación.

Digi International ha desarrollado además un software propio para configurar dichos dispositivos de manera muy intuitiva y mandar datos de un dispositivo a otro. La aplicación se llama *XCTU* [32] (Figura 3-4), y en ella se puede configurar parámetros como la velocidad de transmisión de datos en baudios, el canal de comunicación, la potencia de transmisión, además se puede activar el cifrado de los datos mediante una clave de cifrado AES, etc. El precio de este dispositivo es de 35€.

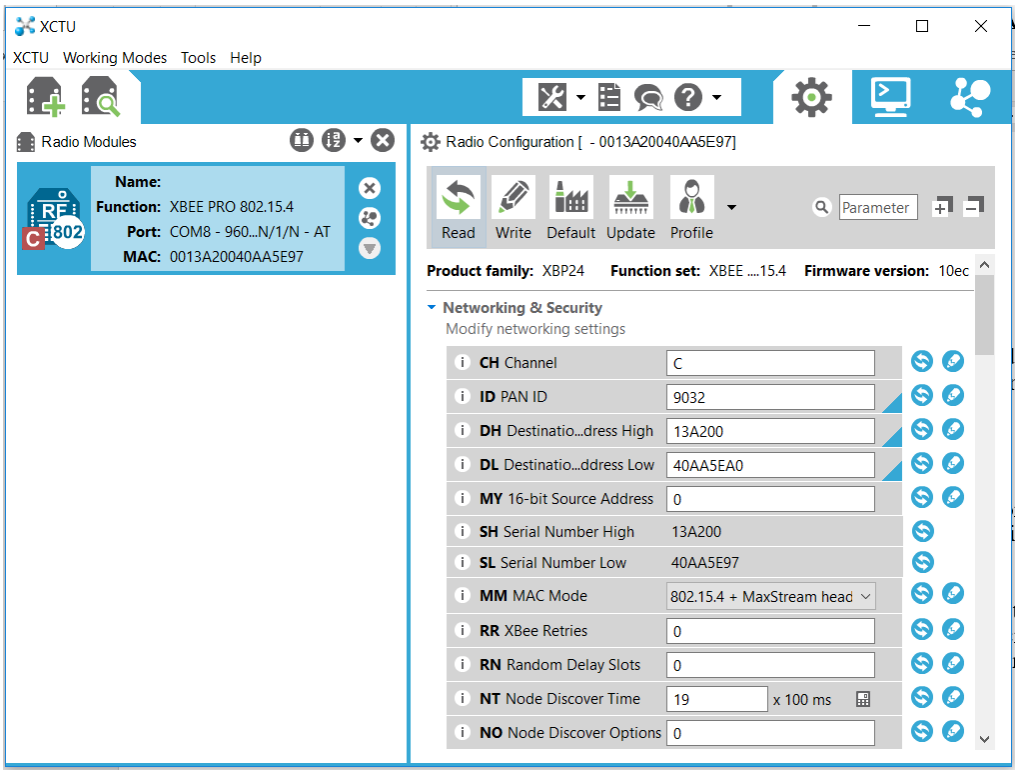


Figura 3-4: Software XCTU.

El motivo por el cual se ha elegido este dispositivo en lugar de los módulos Zigbee con microprocesador integrado del fabricante *Silicon Labs*, fue la sencillez de utilización, ya que ambos cumplían los requisitos del proyecto totalmente. El dispositivo de *Silicon Labs*, al tener integrado un microprocesador reducía considerablemente el tamaño del PCB, pero fue descartado en este primer prototipo por ser un dispositivo nuevo del que se dispone de muy poca documentación.

3.2.2.2 Microprocesador.

El chip elegido es el *ATMEGA32U4* [28], un chip de la marca comercial *Atmel* que cumple con todas las exigencias previas y que además incluye características extras que, en determinados momentos, pueden resultar muy útiles e interesantes.

Además, para facilitar el diseño, se ha seleccionado una plataforma de desarrollo, la placa Teensy *2.0* (Figura 3-5), la cual lleva incorporado dicho chip, además de un reloj de cristal externo a 16MHz, un puerto mini-USB para la programación del chip, un botón de Reset y un LED conectado a uno de los puertos de salida del microprocesador.

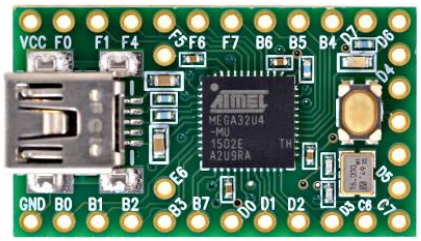


Figura 3-5: Plataforma de desarrollo Teensy 2.0

A continuación, se analizan punto por punto las características imprescindibles:

- Microcontrolador de 8 bits de bajo consumo con 32 kBytes de memoria flash ISP (In-System Self-Programmable) y 2.5 kBytes de memoria SRAM (Static Random Access Memory).
- Voltaje de alimentación ente 2,7 V y 5,5 V.
- Frecuencia de reloj de hasta 16 MHz con cristal externo (incorporado en la plataforma de desarrollo) y alimentación a 5 V u 8 MHz con solo reloj interno.
- USB 2.0 completo.
- Interfaz TWI (Two Wire Interface).
- USART (universal synchronous/asynchronous receiver/transmitter) serie programable.

Además, este microprocesador dispone de las siguientes características complementarias, las cuales podrían usarse trabajos futuros:

- Interfaz SPI maestro/esclavo.
- 26 líneas programables de entrada/salida.
- Gestor de interrupciones tanto internas como externas.
- Watchdog Timer programable.
- Interfaz JTAG (Joint Test Action Group).
- 12 canales de 10 bits para el conversor analógico/digital.

Se ha seleccionado este dispositivo, ya que como se puede observar tiene una velocidad de procesamiento más que suficiente, e implementa los dos buses de comunicación necesarios, el bus I2C, denominado como TWI, y el bus USART, para comunicar con los drivers I2C y con el Xbee respectivamente.

El uso de la plataforma Teensy 2.0, además de proporcionar facilidad de conexión en un tamaño muy reducido, proporciona una plataforma de programación muy sencilla, así como la capacidad de poder ser programada tanto en lenguaje C como en lenguaje Arduino. Cabe destacar que el uso del lenguaje Arduino para la programación del microprocesador, limita considerablemente las capacidades de este, por lo que es mucha mejor opción realizar la programación en lenguaje C.

Para la programación se usará el software propietario de *Atmel*, el *Atmel Studio 7.0* [33]. Este software compila el código C y genera una serie de archivos, entre los que se encuentra el archivo "*.HEX*" que es el encargado de portar el programa de usuario listo para el microprocesador. Este archivo .HEX, se cargará al microprocesador mediante el software proporcionado por Teensy, la cual se llama Teensy Loader (Figura 3-6), que ejecuta el gestor de arranque HalfKay y ejecuta el ultimo .HEX cargado en la placa.

Figura 3-6: Software Teensy Loader.

Como puede observarse, este microprocesador contiene todas las características necesarias para este TFM, además otras características adicionales. Por otro lado, al estar montado en una plataforma de desarrollo, la cual tiene unas dimensiones de 1.6cm x 3cm, facilita considerablemente tanto su montaje como su posterior programación. El precio de esta plataforma de desarrollo es de 16€.

El motivo por el cual se ha elegido este microprocesador en lugar del BeagleBone Pocket, es que, aunque este segundo microprocesador es más potente, no hay gran diferencia en el precio (25€ del BeagleBone frente a 16€ del Teensy 2.0) y tiene implementadas las interfaces necesarias en más de un puerto, no es necesario, ya que con la potencia que proporciona el Teensy 2.0 es suficiente para este proyecto. Además, debido a esta baja potencia, el consumo es considerablemente más bajo, algo que, como este prototipo irá alimentado mediante baterías, es de gran importancia.

3.2.2.3 Driver I2C.

El chip elegido es el PCA9685 del fabricante *NXP* [30]. Este chip es un controlador de 16 canales controlado por bus I2C. Cada salida dispone de un control de PWM individual, operando todas a una frecuencia fija programable entre 24Hz y 1526Hz.

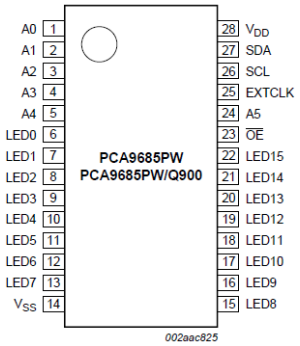


Figura 3-7: Chip PCA9685

Las salidas del chip son de tipo colector abierto con una capacidad de corriente de 25mA a 5V. Tiene un rango de voltaje de alimentación entre 2.3V y 5.5V. Las salidas pueden conectarse directamente a un actuador o a un controlador externo para proporcionar una mayor corriente a los actuadores. Su precio es de 2€.

Estas son sus características principales y de mayor interés para este TFM:

- Permite tiempos de encendido y apagado escalonados para minimizar las corrientes. El retardo de encendido y apagado se puede programar independientemente para cada uno de los 16 canales.
- Tiene 12 bits de resolución de PWM, lo que proporciona 4096 pasos de control para cada salida independientemente.
- Dispone de 6 bits de dirección configurables mediante hardware, lo que permite el uso 64 chips conectados al mismo bus I2C.
- Tiene una entrada de reloj externa (máximo 50MHz) para permitir la sincronización con múltiples dispositivos.
- La corriente de salida permitida es de 25mA en cada puerto de salida.
- Tiene un oscilador interno para el control de la frecuencia del PWM configurable desde 24Hz hasta 1526Hz.

El chip DRV8830 fue descartado, ya que únicamente tiene dos salidas configurables con PWM, por lo que harían falta 64 drives diferentes para el control de los 128 motores necesarios, y solo tiene 3 bits de dirección configurables por Hardware, y por lo tanto únicamente se pueden conectar 9 chips diferentes en el mismo bus I2C.

3.2.3 Protocolos de comunicación.

3.2.3.1 Protocolo I2C.

El protocolo I2C [34] es un protocolo de comunicación basado en 2 hilos, uno encargado de la transmisión serie de los datos, al cual se denomina SDA, y otro para la transmisión del reloj de sincronización, denominado SCL. Estos dos hilos portan toda la información necesaria para establecer comunicación entre dos dispositivos conectados al bus.

En una comunicación mediante este bus, pueden distinguirse dos tipos de dispositivos, a los cuales denominan dispositivos Maestros y dispositivos Esclavos. Se conoce como dispositivo Maestro aquel encargado de iniciar la comunicación, además de enviar y solicitar datos a uno o varios dispositivos esclavos. El dispositivo Maestro también es el encargado de generar la señal de sincronismo o reloj. Los dispositivos Esclavos deben tener una dirección única en el bus, de manera que no haya ningún conflicto en la comunicación.

Las líneas de comunicación SDA y SCL son líneas de comunicación bidireccionales que conectan todos los dispositivos. Además, estas líneas requieren unas resistencias de *pull-up* conectadas a la alimentación (Vdd), de forma que, como los dispositivos conectados a estas líneas, son de tipo colector abierto, se crea un circuito AND (Figura 3-8).

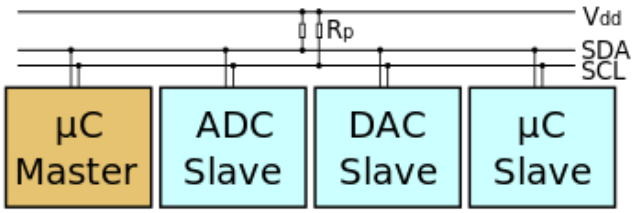


Figura 3-8: Bus I2C con resistencias de pull-up.

Por otra parte, para que una señal se considere como nivel alto (es decir, un "1" lógico), esta debe alcanzar al menos el 70% del voltaje de la fuente que alimenta el sistema (Vdd), y para considerarlo como nivel bajo (es decir, un "0" lógico), debe estar por debajo del 30%.

En el bus I2C, toda comunicación se inicia con una "Condición de Inicio (S)" y se finaliza con una "Condición de Parada", las cuales son generadas por el dispositivo maestro (Figura 3-9). La transmisión se realiza byte a byte seguido por un bit de asentimiento. No hay límites de bytes a transmitir y siempre se comienza la transmisión por el bit más significativo.

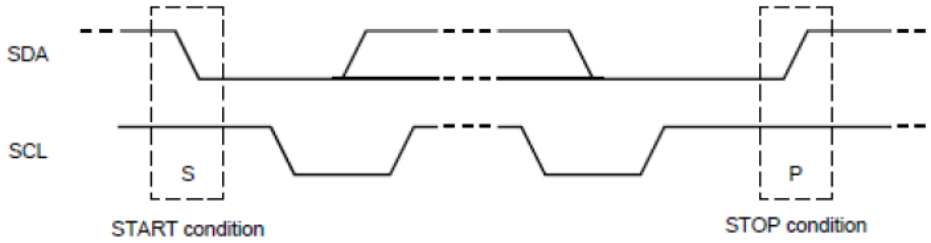


Figura 3-9: Condiciones de START y STOP.

Los bits de asentimiento (ACK o *ACKnowledgement*) o no asentimiento (NACK o *Negative ACKnowledgement*) que siguen a cada byte transmitido sirven para indicar el estado de la

comunicación, ya sea para confirmar que un byte se ha recibido correctamente o incorrectamente, además de para indicar problemas en la transmisión o que alguno de los dispositivos no puede continuar.

Todas las comunicaciones I2C siguen la misma estructura. El primer byte enviado a continuación de la condición de inicio, corresponde con la dirección del esclavo con el que se quiere realizar la comunicación. De esta forma, los 7 primeros bits corresponden a la dirección (ADDR) y el ultimo bit indica si se trata de una operación de lectura (R) o escritura (W). Si la dirección enviada ha sido recibida con el esclavo con dicha dirección, el siguiente bit, correspondería con un asentimiento o ACK. A partir de este momento, el maestro enviará o recibirá los datos, hasta el momento en que se envíe una condición de parada (Figura 3-10).

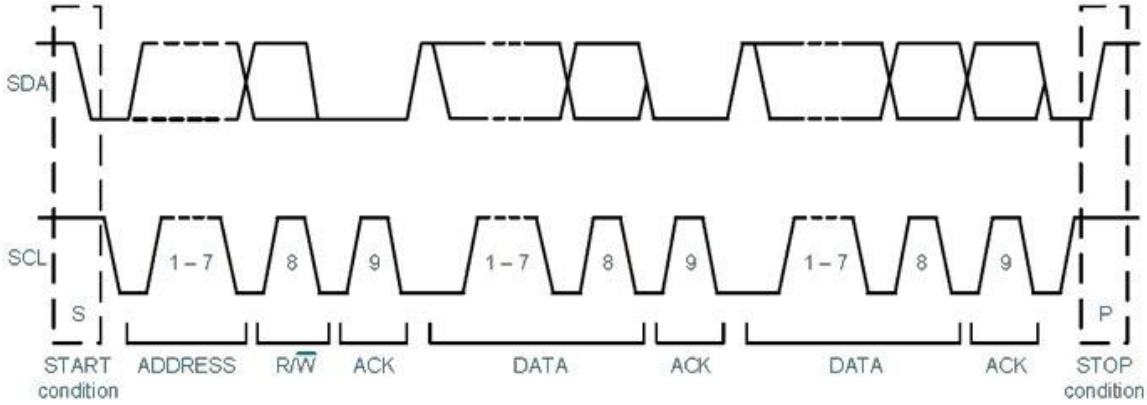


Figura 3-10: Estructura comunicación I2C.

3.2.3.2 USART.

El módulo USART [35] (*Universal Syncronos and Asyncronos Receiver and Transmitter*) consiste básicamente en un puerto serie. Un USART es un módulo de comunicación digital que cuenta con dos conexiones, Rx y Tx, por lo que existen tres modos de comunicación, que pueden ser *Full-Duplex, Half-Duplex* y *Simplex.*

En una comunicación *Full-Duplex*, se pueden recibir y enviar información de manera simultánea, mientras que en una comunicación *Duplex*, o solo transmite o solo recibe información. Por último, en una comunicación *Simplex*, solo se puede transmitir información binaria.

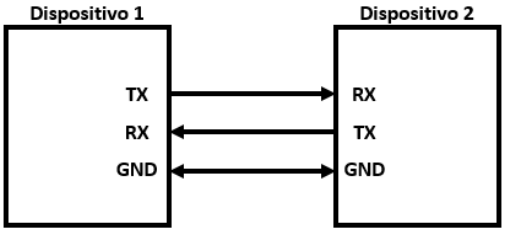


Figura 3-11: Esquema comunicación USART.

La funcion principal del USART es convertir los datos serie a paralelos cuando se trata de los datos recibidos o entrada, y de convertir los datos paralelos a serie para la transmision o datos de salida.

Para que exista una comunicación correcta, es necesario definir una serie de parametros en ambos lados de la comunicación:

- **Sincronismo entre receptro y transmisor:** La sincronización en la transmisión de los datos se lleva a cabo colocando en primer lugar un bit de comienzo (*Start Bit*), después se envían los datos, normalmente entre 5 y 9 bits, y por último se envía un bit de parada (*Stop Bit*). Los USART's que trabajan con 8 bits, añaden un bit de detección de error o bit de paridad.
- **Codificación de los datos:** La codificación puede ser cualquier código binario. El más utilizado es el código ASCII.
- **Prioridad en el envío de los bits.**
- **Tasa de envio de datos:** La tasa de transmisión se mide en baudios o número de bits que se transmiten por segundo (bps). En principio puede ser cualquier valor, pero las tasas estándares son: 110, 150, 300, 600, 900, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400 y 57800, aunque la más utilizada es la de 9600 bps. La tasa de baudios viene dada como:

$$baudios = \frac{f_i}{16 * (n + 1)}$$

  Donde f_i es la frecuencia de reloj y n puede ser cualquier valor entre 00_{16} y FF_{16}.

- **Señales *Handshaking.***
- **Señales eléctricas de los valores lógicos.**

3.2.4 Solución propuesta.

Una vez vistos los principales dispositivos a utilizar, las tecnologías que implementan y los protocolos de comunicación que mejor se adaptan al problema que presenta este proyecto se puede proponer una solución que satisfaga los objetivos del mismo. A continuación, se muestra un diagrama de bloques que explica de forma sencilla los principales detalles de esta solución:

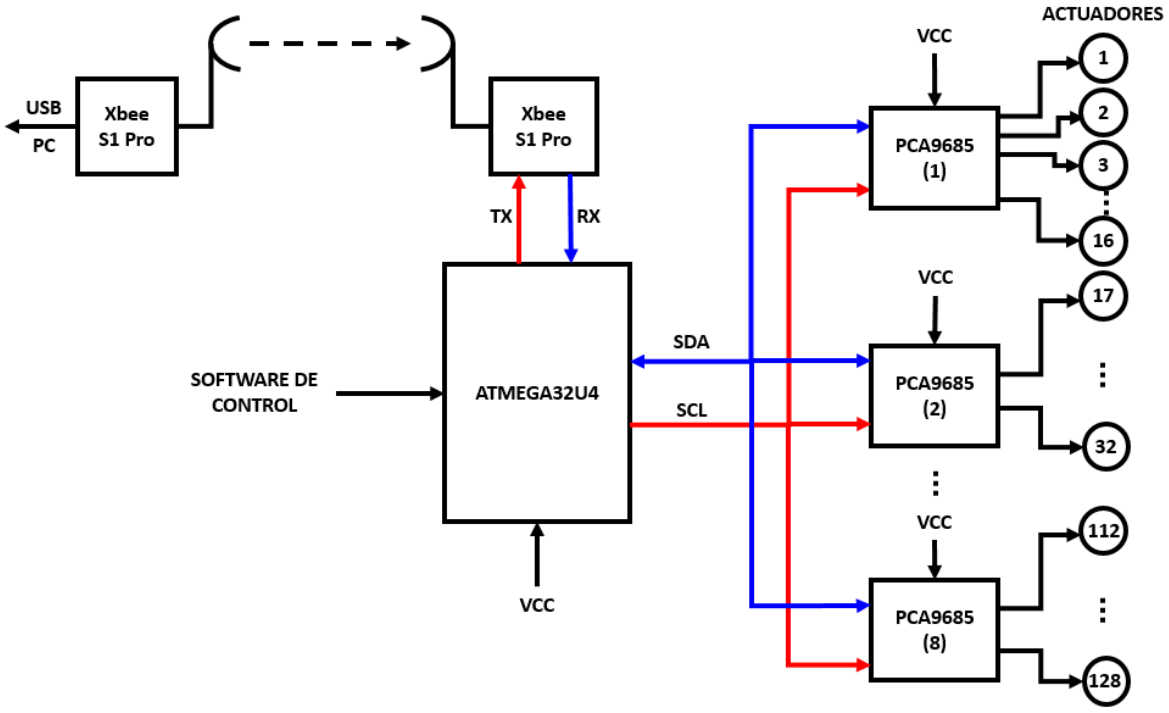


Figura 3-12: Diagrama de la solución propuesta.

La solución que se propone son varios PCBs (*Printed Circuited Board*) con las dimensiones más reducidas posibles, los cuales se organizan de la siguiente forma:

- Un PCB que contenga el microprocesador (ATMEGA32U4) junto con el dispositivo de comunicación inalámbrica (Xbee) de forma que se pueda recibir la información enviada por el dispositivo Xbee conectado al ordenador donde se procesan las imágenes. Además, incorporará un conector para poder llevar un cable el bus I2C a los diferentes drivers controladores.

- Uno o varios PCBs que contengas los drivers I2C (PCA9685) junto con unas pequeñas conexiones para los motores de vibración, junto con otro conector para llevar el cable con el bus I2C desde el microprocesador.

4 Desarrollo del Prototipo

4.1 Introducción.

El desarrollo del prototipo, tanto a nivel de hardware como a nivel de software, se ha divido en múltiples etapas. En las primeras etapas llevadas a cabo, se ha creado y desarrollado las plataformas hardware necesarias para la creación del prototipo y desarrollo de librerías software. En una primera etapa, se ha creado en una herramienta de desarrollo un PCB que contendría el microprocesador junto con el chip Xbee, además de un conector para el bus I2C, y un conector de alimentación (Módulo 1: Comunicaciones y Control en la Figura 4-1). Seguidamente se desarrolló un segundo PCB de pruebas que contendría el driver I2C junto con nuevamente conector para el bus I2C y otro conector para los 16 motores (Módulo 2: Drivers y Actuadores en la Figura 4-1).

Una vez creadas los PCBs de pruebas, se pasó a una siguiente etapa de creación y desarrollo de las librerías de control de los diferentes protocolos que irán cargadas en el microprocesador. Para ello se ha investigado y desarrollado la librería necesaria para el uso del bus I2C disponible en el microprocesador. De igual manera, se ha desarrollado una librería para el uso del bus USART.

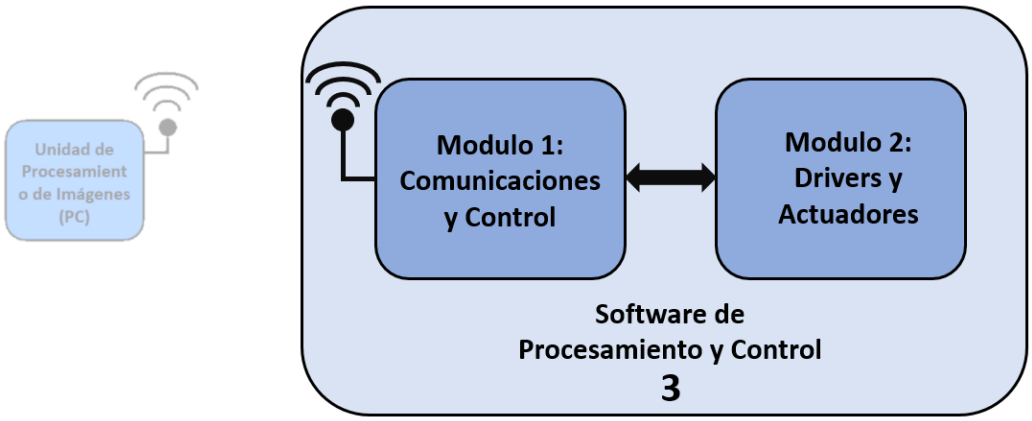


Figura 4-1: Esquema del prototipo a alto nivel.

Una vez que las librerías estaban completas y funcionando, y todos los elementos hardware fabricados, se desarrolló el software de control del microprocesador, el cual sería el encargado del funcionamiento del prototipo al completo.

Por último, en una etapa final, debido a que el PCB del driver I2C fabricado era un diseño de pruebas, se desarrolló un PCB final para el prototipo.

Es importante destacar, que, durante todas las etapas de desarrollo del prototipo, se realizaban pequeños fragmentos de código para comprobar el funcionamiento de cada elemento de forma individual, además de así investigar y aprender la programación y control de cada elemento.

A continuación, se van a detallar cada una de las etapas llevadas a cabo durante el desarrollo.

4.2 Desarrollo Hardware.

4.2.1 Módulo 1: Comunicaciones y Control.

Esta primera etapa se centró principalmente en el diseño de PCBs para realizar la interconexión de los dispositivos de comunicaciones y control.

En primer lugar, se diseñó el PCB que contendría a la plataforma Teensy 2.0 y el dispositivo de comunicaciones Xbee, de forma que, mediante un conector de entrada, se alimentasen y comunicasen entre ellos. Además, también contendría un conector que conectase con el bus TWI proporcionado por el microprocesador.

Esta placa PCB, se ha decidido que en lugar de soldar directamente los componentes (Teensy 2.0 y Xbee), se colocaran unos conectores de forma que, si durante el uso del prototipo, se rompe alguno de los dispositivos, sean fácilmente sustituibles, o incluso en el caso del Xbee, si se desea utilizar un modelo superior al seleccionado, pueda sustituirse sin problemas, ya que en la mayoría de los modelos se mantiene el mismo encapsulado. Para realizar estos diseños, se utilizó el software *Altium Designer*.

Altium Designer (Figura 4-2) es un conjunto de herramientas y programas para el diseño electrónico en todas sus fases. Con él se realizará la parte del diseño de esquemático, creación y rutado del PCB, testeo de errores y, por último, generación de los archivos Gerber y de taladros para la máquina de fabricado de PCBs.

Es importante destacar que, previo a la realización del diseño de los PCBs, se realizó un pequeño estudio e investigación sobre el funcionamiento y control del Software, ya que era prácticamente desconocido para mí.

En un primer lugar, para diseñar el PCB, se tiene que realizar el esquemático en el que se conectaran todos los dispositivos que formaran dicho PCB. Para ello, se estudió en profundidad las hojas de características de los chips principales y notas de aplicación, prestando particular atención a todos los elementos que rodean a estos chips y sin los cuales no podrían funcionar.

Una vez hecho esto, se puede comenzar a diseñar el esquemático. Para ello, se crea un nuevo proyecto del tipo "*PCB Project*", y se añade una nueva pestaña de esquemático sobre la que se irán colocando los diferentes chips que formaran el PCB y las conexiones necesarias.

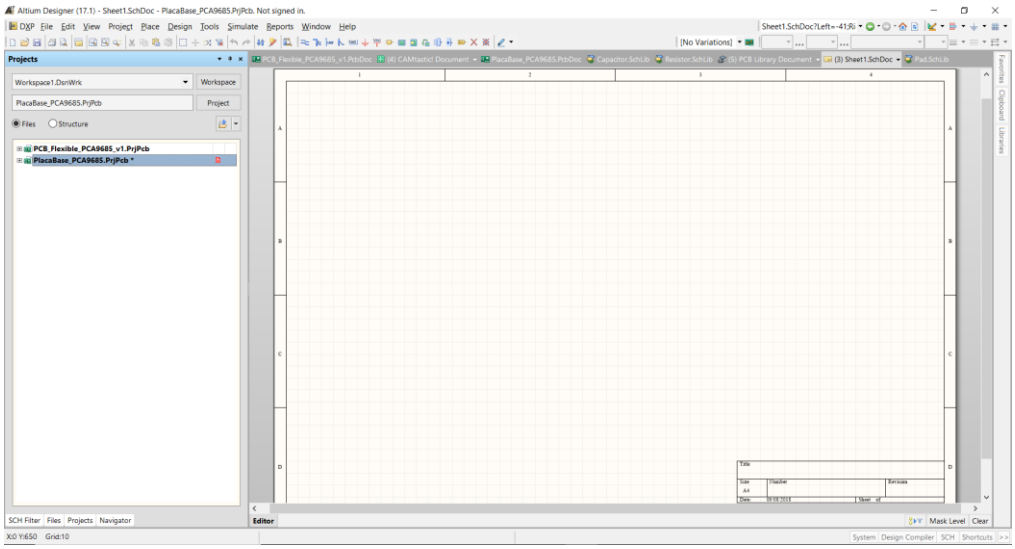


Figura 4-2: Interfaz de *Altium Designer* para el editor de esquemáticos.

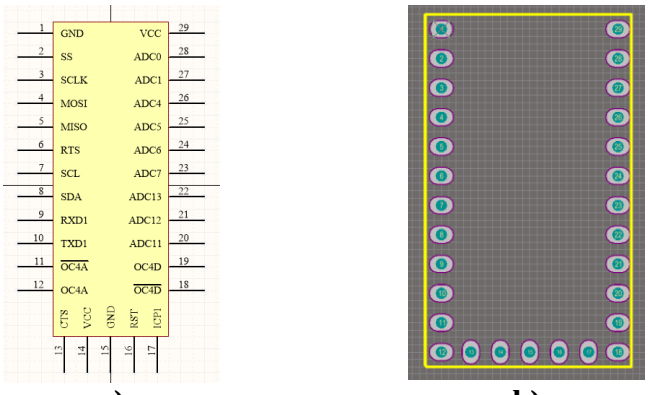
Los componentes más estandarizados y típicos, como pueden ser las resistencias, condensadores, etc, se pueden encontrar en las librerías disponibles en el software, de forma que aquellos que no se encuentren, habría que créalos desde cero, tanto el esquemático como el *footprint*.

Por ello, se tienen que crear las librerías correspondientes del zócalo para la plataforma Teensy 2.0 y el Xbee Pro S1, así como el conector de 4 pines.

4.2.1.1 Esquemático y Footprint de la plataforma Teensy 2.0.

Tal y como se ha indicado antes, se ha decidido que, en lugar de conectar directamente este dispositivo al PCB, se colocaran unos conectores hembra para formar un zócalo con las dimensiones y conexiones necesarias para este componente.

Para ello, el esquemático creado para este dispositivo (Figura 4-3 a)), consta de 29 pines, de los cuales únicamente se usarán los pines Vcc y GND (para alimentación y tierra), SCL y SDA (para el bus TWI), y RXD1 y TXD1 (para la conexión del módulo USART al Xbee).



a) **b)**

Figura 4-3: Esquemático y Footprint de Teensy 2.0.

De la misma manera, como en el PCB únicamente se colocarán conectores hembra donde se pondrá el Teensy 2.0, el footprint (Figura 4-3 b)), como se puede observar, es muy sencillo, ya que únicamente es necesario colocar los diferentes pads para los conectores con una separación entre pines de 2.54mm.

4.2.1.2 Esquemático y Footprint del módulo Xbee Pro S1.

En segundo lugar, se tiene que crear la librería con el esquemático y footprint del módulo Xbee Pro S1.

Para este módulo se seguirá la misma metodología que en la librería anterior (4.2.1.1), ya que únicamente se colocaran unos conectores hembra donde se conectará posteriormente el Xbee. En este caso, el dispositivo cuenta con 20 pines, 10 a cada lado, de los cuales, únicamente se usarán los pines Vcc y Gnd (para la alimentación y tierra), y Data In y Data Out (que componen los pines del puerto serie que se conectará con el módulo USART del microprocesador).

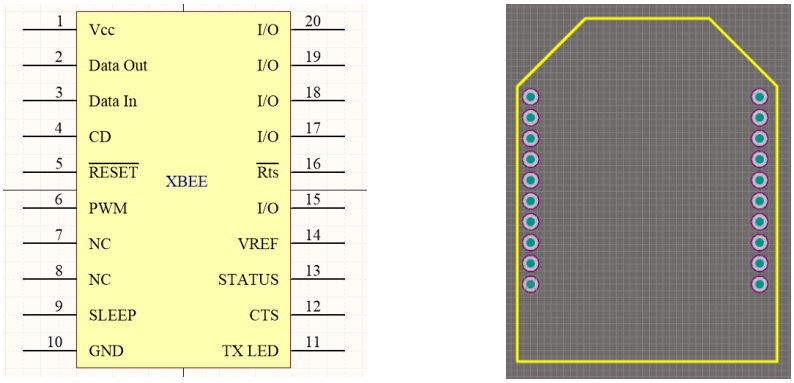


Figura 4-4: Esquemático y footprint del módulo Xbee.

En este caso, como la forma del módulo Xbee no es rectángular, el contorno amarillo que se puede observar en el footprint (Figura 4-4), marca el área que será ocupada por este chip, se forma que no se puede colocar ningún otro componente debajo que interfiera en su conexión. Cabe destacar, que los conectores hembra que se usarán para este módulo, no son los típicos con un paso de 2.54mm, si no que serán necesarios conectores con paso de 2mm.

4.2.1.3 Alimentación del PCB.

Como este PCB será el encargado de controlar todo el prototipo y no irá conectado a nada más, es necesario colocar un módulo para alimentación de los diferentes dispositivos.

Para ello, en primer lugar, se colocará un conector de alimentación hembra (Figura 4-5), donde o bien se conectarán unas baterías, o se conectará a la corriente eléctrica con un adaptador, por lo que, si se conecta al PCB una pila de 9V, el voltaje de alimentación obtenido será demasiado alto para los componentes del PCB.

Figura 4-5: Conector de alimentación hembra.

Analizando las hojas de características de los dos componentes principales de este PCB, Teensy 2.0 y Xbee Pro S1, los voltajes de alimentación recomendados por los fabricantes son, entre 2.7V y 5.5V para le Teensy 2.0, y entre 2.8V y 3.4V para el módulo Xbee, por lo que se podría unificar la alimentación a una sola fuente de 3.3V, la cual se conseguirá conectado al conector de alimentación, un regulador de tensión a 3.3V. Adicionalmente, si se mira la hoja de características del dispositivo PCA9685 montado en el PCB anterior (4.3.3), acepta la tensión de alimentación de 3.3V, por lo que con esta única fuente, se podría alimentar todo el prototipo. El regulador elegido para tal fin ha sido el regulador REG104 (Figura 4-6).

Figura 4-6: Regulador de tensión a 3.3V REG104.

Si se analiza detalladamente la hoja de componentes de este último dispositivo, se puede ver que el fabricante recomienda realizar el siguiente esquema de conexiones (Figura 4-7), de manera que al pin de entrada Vin, vaya conectado el pin Vcc del conector de alimentación anterior, y así se obtenga en Vout el voltaje de 3.3V necesario para alimentar el resto de componentes.

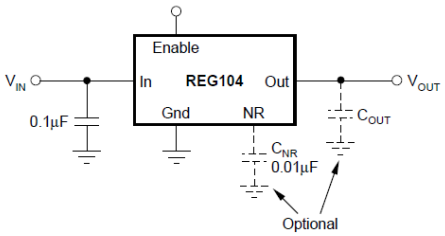
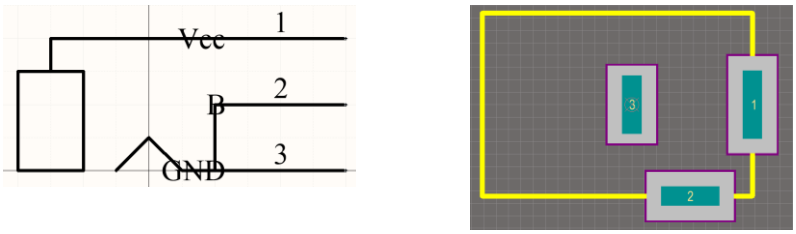


Figura 4-7: Esquema de conexión del regulador REG104.

De igual forma a todos los componentes anteriores, en el software *Altium*, no se dispone de las librerías necesarias para ambos componentes, por lo que será necesario crearlas.

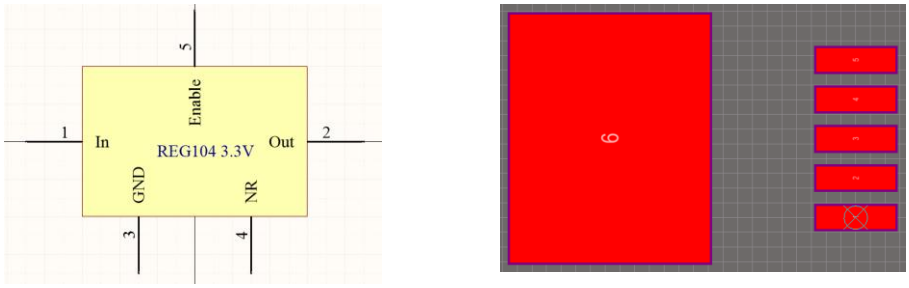
El conector de alimentación únicamente consta de 3 pines, Vcc, Gnd y un conector auxiliar para coger la alimentación de otro punto en el caso de que no haya nada conectado a este conector. Por lo tanto, el esquemático y footprint de este componente son:



a) **b)**

Figura 4-8: Esquematico y Footprint del conector hembra de alimentación.

El regulador de tensión está formado por 5 pines de montaje superficial o SMD, además de esos 5 pines, el fabricante recomienda colocar un área de cobre debajo del componente para la disipación de calor. Por lo tanto, el esquemático y footprint de este componente quedarían de la siguiente forma:



a) **b)**

Figura 4-9: Esquemático y Footprint del Regulador REG104.

4.2.1.4 Esquemático y Layout del PCB completo.

Una vez desarrolladas todas las librerías necesarias, ya se pueden colocar todos los dispositivos en el esquemático y realizar las conexiones correspondientes. Al igual que en el PCB anterior, el esquemático es muy sencillo, y prácticamente pueden hacerse las conexiones de forma directa.

En el caso del dispositivo Teensy 2.0, al ser una plataforma de desarrollo que contiene al microprocesador ATMEGA32U4, ya tiene todas las conexiones necesarias realizadas para un correcto funcionamiento del micro, únicamente se añadirá un condensador de desacoplo junto al pin de alimentación.

En cambio, para el dispositivo Xbee, aunque se podría conectar directamente, tal y como se puede ver en la hoja de características del dispositivo, el fabricante recomienda colocar dos condensadores de desacoplo, de 1µF y 8.2pF, en el puerto de alimentación, con el objetivo de eliminar el ruido.

Por último, al igual que en el PCB anterior, se hará uso del conector de 4 pines, por el cual se llevará el bus TWI que conectará el microprocesador con el driver I2C del otro PCB. Esta librería, como ya ha sido desarrollada para el PCB anterior, puede reutilizarse en este nuevo PCB.

Por lo tanto, una vez conectados todos los puertos de cada uno de los componentes, se obtiene el siguiente esquemático:

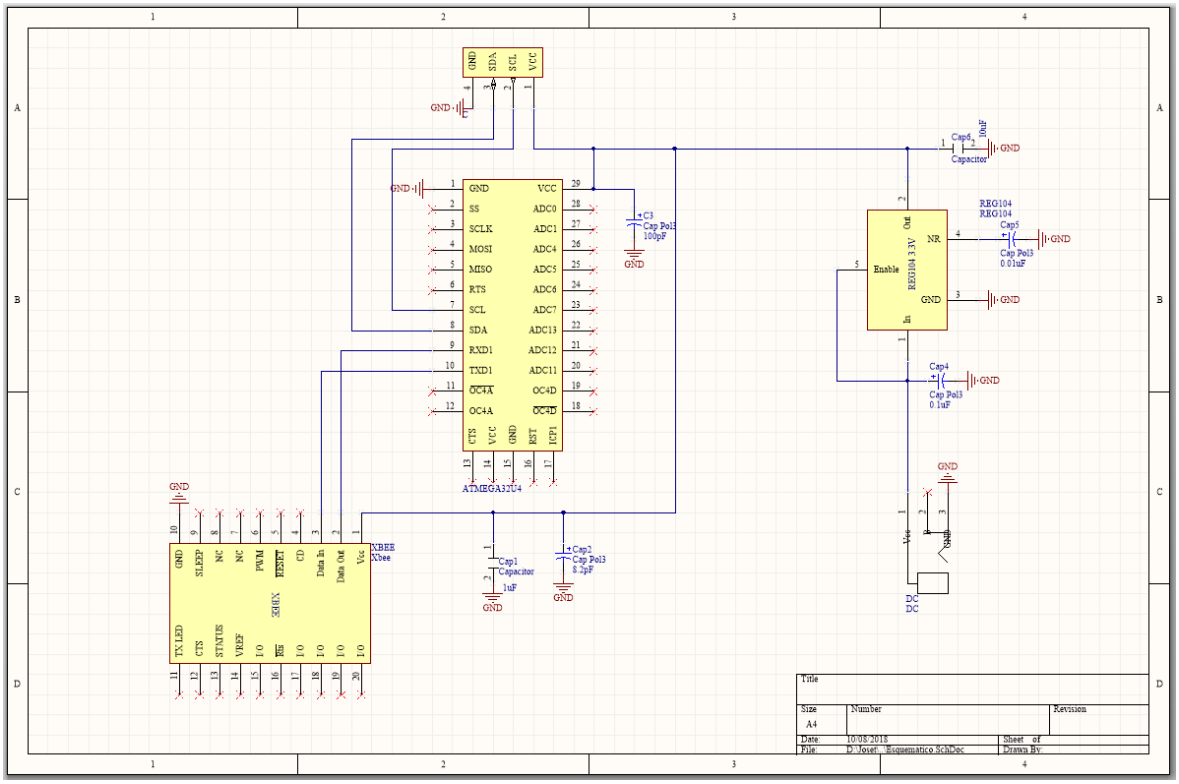


Figura 4-10: Esquemático PCB Microprocesador.

Una vez finalizado el esquemático, se desarrollará el layout del PCB. Nuevamente, se mantendrá una separación entre pistas y ancho de pistas de 0.3mm. En cambio, este nuevo PCB, se va a realizar en un PCB de una cara, donde todas las pistas estarán en la cara inferior, y los componentes irán en montados en la cara superior, salvo los componentes de montaje superficial, de forma que el ensamblado de los componentes al PCB sea sencillo.

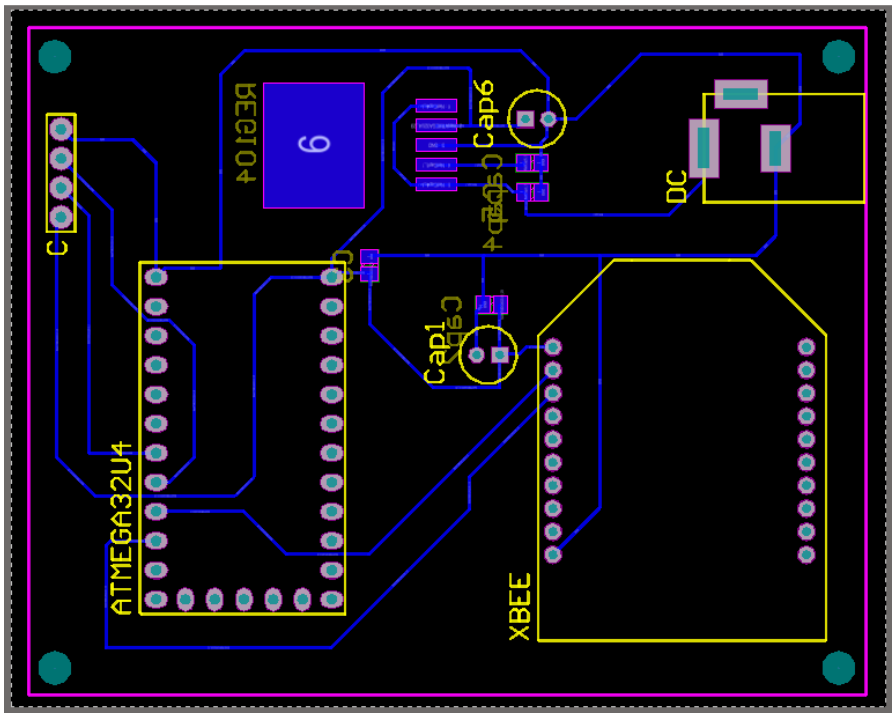


Figura 4-11: PCB del Microprocesador y el módulo Xbee.

4.2.1.5 Fabricación PCB y soldado de componentes.

Por último, una vez el PCB está correctamente diseñado, se exportan los ficheros GERBER, los cuales llevan los diseños de cada una de las caras y del contorno del PCB, y un fichero con la posición y tamaño de los agujeros. Estos ficheros se introducirán en la máquina de fabricado de PCBs, la cual realizará las pistas y taladros en una placa de una cara, para posteriormente soldar cada uno de los componentes.

Una vez terminados todos los procesos, se obtiene el siguiente PCB (Figura 4-12 y Figura 4-13):

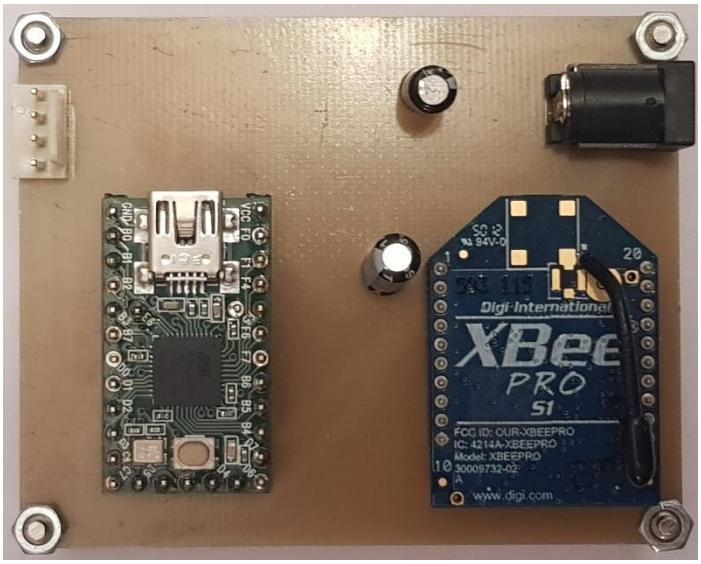


Figura 4-12: PCB del microprocesador y Xbee – Cara superior.

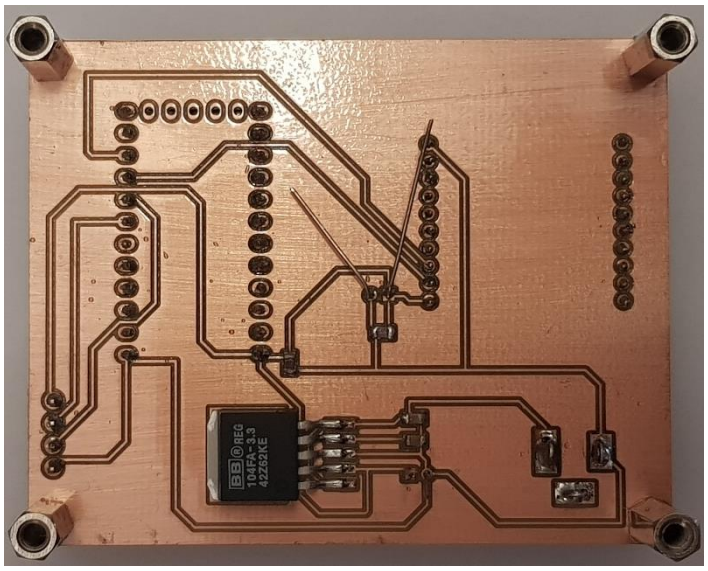


Figura 4-13: PCB del microprocesador y Xbee – Cara inferior.

4.2.2 Módulo 2: Driver y Actuadores.

Una vez terminado el PCB que contendría al módulo de comunicaciones y control, es necesario realizar un segundo PCB, el cual contenga al driver PCA9685 y tres conectores, el primero de ellos se utilizará para conectar el bus I2C entre el microprocesador y el driver, y los dos siguientes para la conexión entre el driver y los motores.

Nuevamente, se desarrollará el PCB con *Altium,* donde para comenzar con el diseño de dicho PCB, se necesitan crear las librerías (esquemático y footprint) para los componentes a utilizar.

4.2.2.1 Esquemático y Footprint del PCA9685.

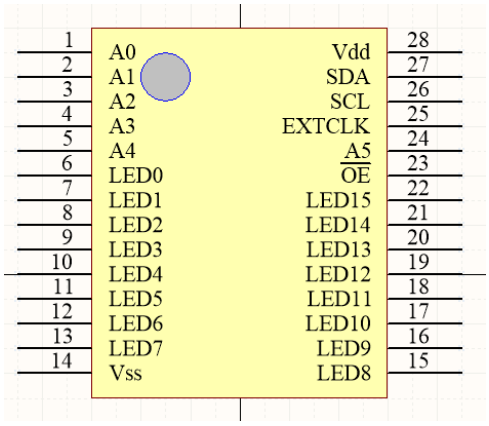


Figura 4-14: Esquemático PCA9685.

El esquemático del dispositivo estará formado por 28 pines. Los 5 primeros pines y el pin 24 (A0, A1, A2, A3, A4 y A5) son los encargados de configurar la dirección Hardware que

tendrá este dispositivo en el bus I2C. Dicha dirección tendrá la siguiente forma (Figura 4-15), y deberá ser única para cada driver conectado al bus:

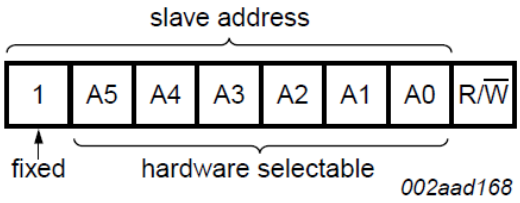
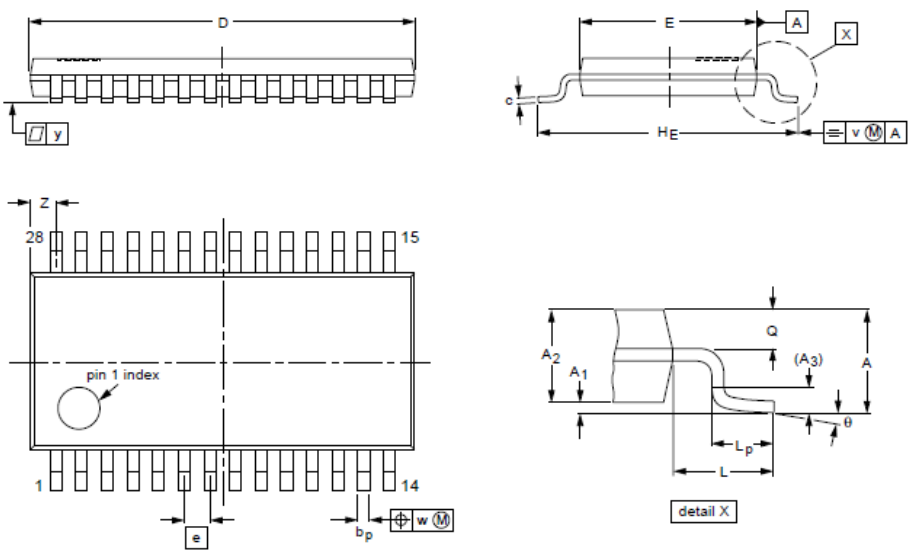


Figura 4-15: Bits de dirección Hardware del dispositivo.

Es importante destacar que algunas direcciones son direcciones reservadas para realizar ciertas funciones, por lo que se tendrían que evitar para un correcto funcionamiento del driver.

El diseño del esquemático es algo muy sencillo y simple, ya que únicamente hay que colocar los pines con sus nombres correspondientes. En cambio, para el desarrollo del *Footprint*, hay que ser más cuidadoso, ya que aquí es necesario realizar la huella del chip con las medidas exactas.

Si se busca en la hoja de características del chip, se puede ver que existen dos tipos de encapsulados diferentes, el TSSOP28 y el HVQFN28. En este caso, únicamente se va a realizar el footprint del encapsulado TSSOP28 (Figura 4-16 y Figura 4-15), ya que es el que se va a usar.



DIMENSIONS (mm are the original dimensions)

UNIT	A max.	A_1	A_2	A_3	b_p	c	$D^{(1)}$	$E^{(2)}$	e	H_E	L	L_p	Q	v	w	y	$Z^{(1)}$	θ
mm	1.1	0.15 0.05	0.95 0.80	0.25	0.30 0.19	0.2 0.1	9.8 9.6	4.5 4.3	0.65	6.6 6.2	1	0.75 0.50	0.4 0.3	0.2	0.13	0.1	0.8 0.5	8^o 0^o

Figura 4-16. Dimensiones PCA9685 TSSOP28.

Por lo tanto, teniendo en cuenta todas las dimensiones de este dispositivo, se ha diseñado el siguiente *Footprint* (Figura 4-17) en *Altium*, donde los números de los *Pads*, se corresponden con los números de los pines del esquemático creado.

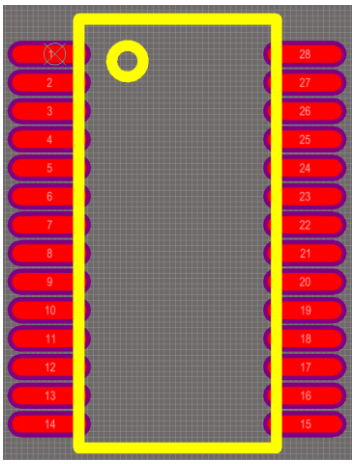
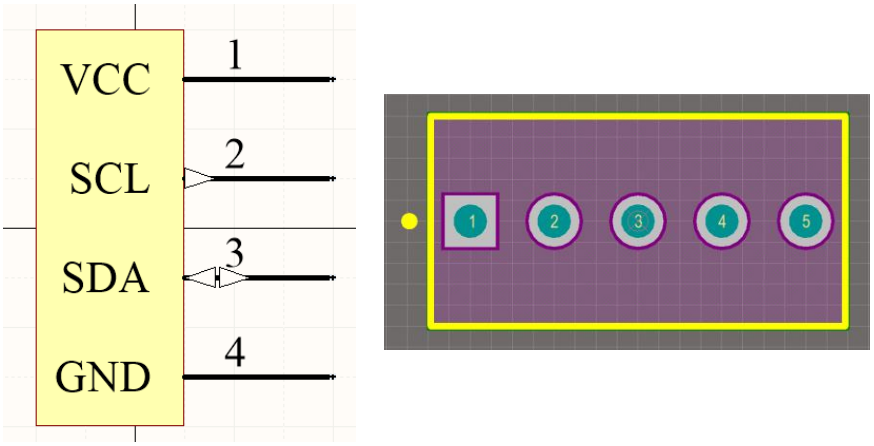


Figura 4-17: Footprint PCA9685 TSSOP82.

4.2.2.2 Esquemático y Footprint de los conectores.

De igual manera, se crearon además los esquemáticos y *Footprints* de los 3 conectores antes mencionados.

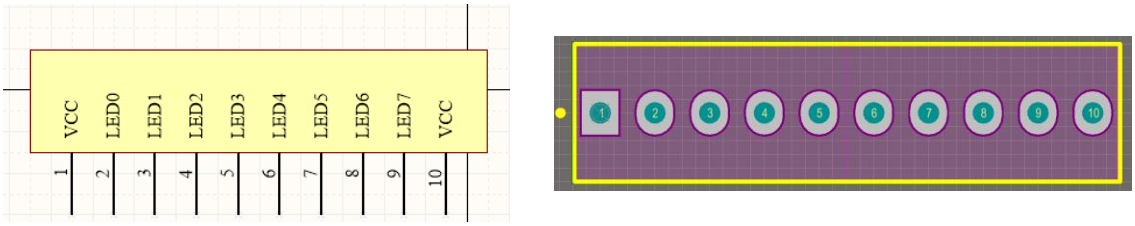
El conector pequeño de 4 pines (Figura 4-18), servirá para conectar un cable desde el microprocesador hasta esta placa, por el cual se transmitirá el bus I2C para el control del driver.



a) **b)**

Figura 4-18: a) Esquemático conector 4 pines. b) Footprint conector 4 pines.

Por otra parte, los dos conectores de 10 pines (Figura 4-19) se van a utilizar para conectar las 16 salidas del driver, a las cuales se conectarán los motores, y los 4 pines restantes para llevar líneas de alimentación.



a) **b)**

Figura 4-19: a) Esquemático conector 10 pines. b) Footprint conector 10 pines.

4.2.2.3 Esquemático y Layout del PCB completo.

Una vez desarrolladas todas las librerías necesarias, ya se pueden colocar todos los dispositivos en el esquemático y realizar las conexiones correspondientes. Como este esquemático es muy sencillo, prácticamente pueden hacerse las conexiones de forma directa, pero es recomendable, que para un funcionamiento correcto del chip PCA9685 se hagan las conexiones recomendadas por el fabricante (Figura 4-20), las cuales pueden encontrarse en la hoja de características de este.

Tal y como se puede ver a continuación, se recomienda colocar resistencias de pull-up en las entradas SDA, SCL y OE.

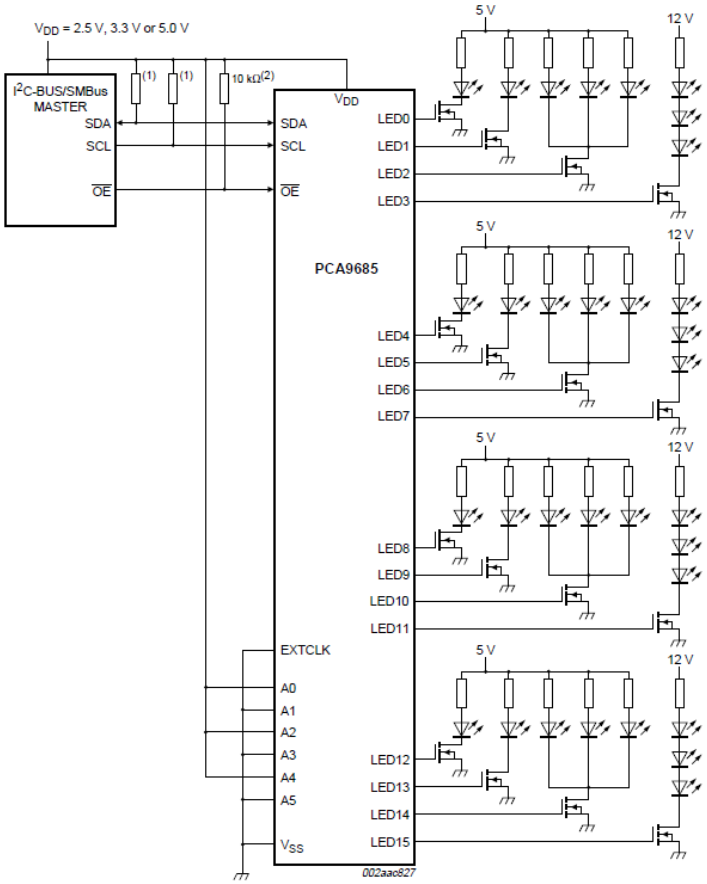
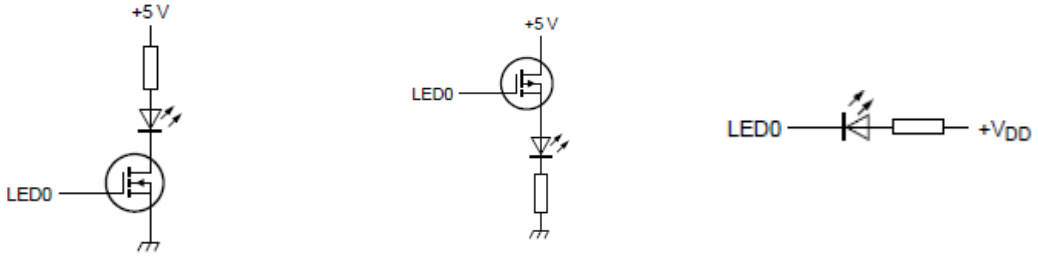


Figura 4-20: Típica aplicación del PCA9685

Además, las salidas del chip que irán conectadas a los motores, se puede ver en la hoja de características, que permiten 3 tipos de conexión (Figura **4-21**), o con un transistor NPN, un transistor PNP o una conexión directa. En este TFM se usará la conexión directa.



a) Transistor NPN **b) Transistor PNP** **c) Forma Directa.**

Figura 4-21**: Tipos de conexión de las salidas del PCA968**

De esta forma, el esquemático resultante sería el siguiente:

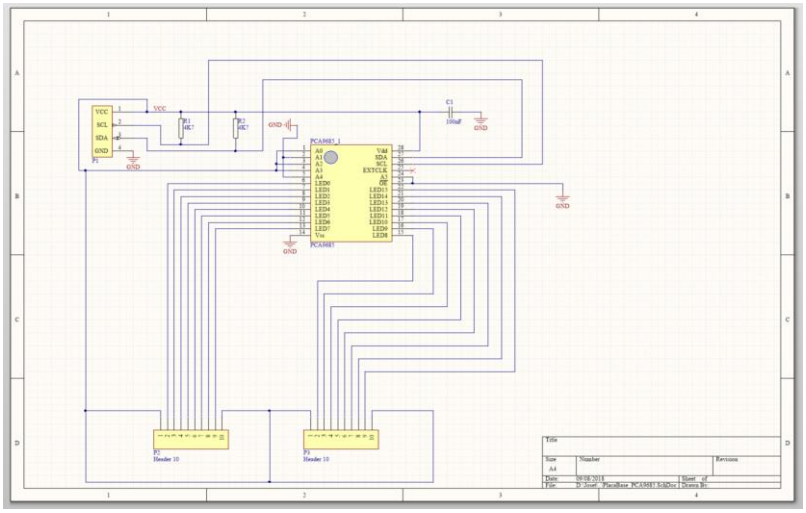


Figura 4-22: Esquemático PCB para el chip PCA9685.

En este esquemático puede observarse que, en primer lugar, la dirección Hardware que se le va a asignar a este PCB es la 0x4D. Además, se han hecho uso de las resistencias de pull-up para las entradas SDA y SCL, y se ha conectado un condensador de desacoplo en la alimentación del chip. Por último, aunque este PCB no llevará soldados los motores, o leds en su defecto, se puede observar que las conexiones que se van a utilizar en la salida, son conexiones directas, sin hacer uso de ningún driver externo.

Una vez se ha terminado el esquemático, únicamente queda crear el Layout del PCB. Para ello, es necesario crear un nuevo fichero de PCB en el cual, en primer lugar, se le asignaran unas dimensiones determinadas, en nuestro caso el PCB medirá 40.64mm x 38.10mm. Además, te permite seleccionar cuantas capas formaran este PCB, en nuestro caso, debido a la tecnología de la que se dispone, solo se pueden hacer PCBs de doble cara, que en este caso es suficiente. Por último, se configurarán que las pistas tengan un grosor de 0.3mm y una separación mínima entre ellas de 0.3mm, y todos los dispositivos irán colocados sobre la cara superior.

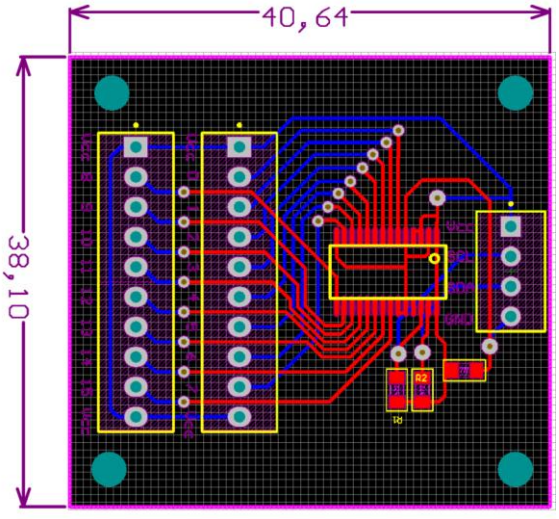


Figura 4-23: PCB del chip PCA9685.

Como se puede observar en el PCB anterior (
Figura **4-23**), se han colocado las pistas en diferentes caras, donde la cara superior se corresponde con el color rojo, y la cara inferior con el color azul.

4.2.2.4 Fabricación PCB y soldado de componentes.

Por último, una vez el PCB está correctamente diseñado, se exportan los ficheros GERBER, los cuales llevan los diseños de cada una de las caras y del contorno del PCB, y un fichero con la posición y tamaño de los agujeros. Estos ficheros se introducirán en la máquina de fabricado de PCBs, la cual realizará las pistas y taladros en una placa de doble cara, para posteriormente soldar cada uno de los componentes.

Una vez terminados todos los procesos, se obtiene el siguiente PCB (Figura 4-24 y Figura 4-25):

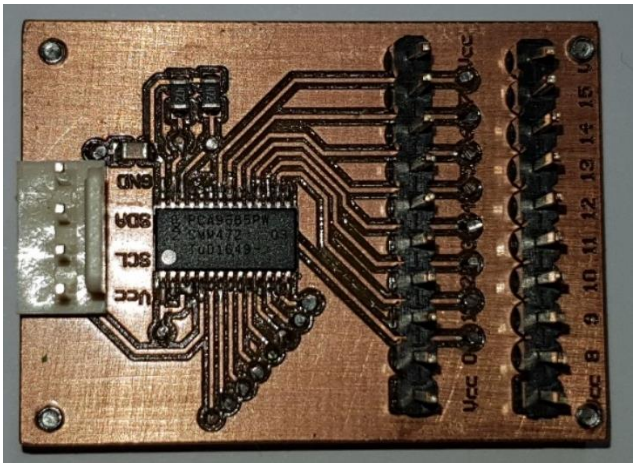


Figura 4-24: PCB del chip PCA9685 – Cara Superior.

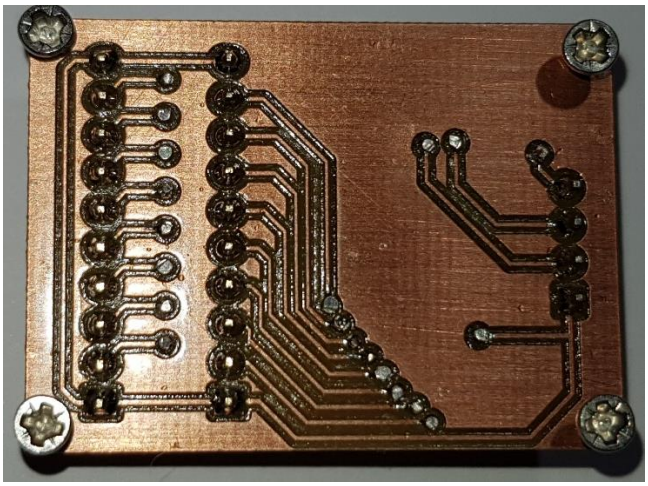


Figura 4-25: PCB del chip PCA9685 – Cara Inferior.

4.3 Desarrollo Software.

4.3.1 Desarrollo librería I2C – TWI.

Uno de los protocolos de comunicación más importante que debe implementar el prototipo desarrollado, tal y como se ha indicado a lo largo de este documento, es el protocolo I2C, el cual se encargará de controlar el estado (encendido, apagado e intensidad de vibración) de cada uno de los motores que formaran este prototipo. Es por ello, que un requisito indispensable que debería tener el microprocesador seleccionado era que tuviese implementado este protocolo.

El microprocesador ATMEGA32U4 tiene implementado un protocolo al cual denominan TWI (*Two Wire Serial Inteface*) [28], el cual es técnicamente el mismo protocolo que el I2C explicado en 3.2.3.1, salvo que por motivos de licencias y problemas de patentes del I2C, el fabricante *Atmel* desarrollo su propio protocolo.

El protocolo TWI es un protocolo orientado al Bytes y basado en interrupciones. Estas interrupciones se emiten después de todos los eventos del bus, como puede ser una condición de START. Debido a esto, el software de la aplicación puede realizar otras operaciones durante una transferencia de bytes TWI.

La configuración y programación del protocolo TWI se hace a través de una lista de registros, de un Byte de tamaño, disponibles en el microprocesador. Cada uno de ellos, en función del valor que tenga en cada uno de sus bits, realizan una función determinada, como puede ser enviar una condición de START. Para ver más información sobre cada uno de los registros y cuál es su función dentro del microprocesador, ver el Anexo A.A.

4.3.1.1 Implementación de la librería TWI.

Una vez se han descrito todos los registros de control del bus TWI y cuáles son sus funciones, se puede continuar con el desarrollo de la librería para poder usar dicho protocolo en el microprocesador. A continuación, se va a analizar función a función la librería desarrollada para tal fin:

- **Void init_i2c().**
  Esta función es la encargada de inicializar los parámetros de bus TWI. Tal y como se puede observar en el fragmento de código siguiente, en primer lugar, se asignan a los puertos D0 y D1 de la tarjeta Teensy 2.0 las señales SDA y SCL. Seguidamente se asigna al registro TWBR (A.A) el factor de división para el factor generador de tasa de bits, donde F_CPU es 16MHz y SCL_CLOCK es 400KHz, obteniendo de esta forma un factor de división de 12. Por último, se pone el valor del Prescaler a 0 (TWSR) y se habilita el bus TWI poniendo a 1 el bit 2 (TWEN) en el registro TWCR (A.A.2).

```
/*************************************************************
/                                                           **
/     Función para iniciar el bus I2C, asignando el puerto **
/     D0 para SCL y D1 para SDA. Asigna también la          **
/     frecuencia, y habilita el bus.                        **
/                                                           **
*************************************************************/
void init_i2c() {
      PORTD &= ~(1 << 0); //Puerto D0 SCL
      PORTD &= ~(1 << 1); //Puerto D1 SDA
      TWBR = (((F_CPU / SCL_CLOCK) - 16) / 2);
      TWSR = 0;
      TWCR = (1 << TWEN); // Habilita el bus TWI
}
```

- **Void start_i2c().**
  Esta función se encarga de enviar una condición de START a través del bus, siempre y cuando el microprocesador actúe como dispositivo maestro. Para ello, en el registro TWCR (A.A.2) se ponen a "1" los bits TWINT, TWSTA y TWEN, los cuales corresponden con los bits 7, 5 y 2 respectivamente. De esta forma, con TWINT a 1 se indica que se ha terminado el trabajo actual, con TWSTA que el dispositivo va a funcionar como maestro y además envía condición de START, y con TWEN se vuele a habilitar el bus. Después, se espera hasta que el bit TWINT este a "0", o lo que es lo mismo, se haya terminado de enviar la condición de START.

```
/*************************************************************
/                                                           **
/      Función para enviar la condición de inicio de        **
/      comunicación.                                        **
/                                                           **
*************************************************************/
void start_i2c() {
       // Envía la condición de inicio
       TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
       //Espera hasta que TWINT se ponga a 0. Esto indica que la
condición de inicio ha sido transmitida.
       while (!(TWCR & (1<<TWINT)));

}
```

- **Void send_Address_i2c(uint8_t ADDR, uint8_t w).**
  Esta función es la encargada de enviar un byte compuesto por la dirección del esclavo (ADDR) seguido de un "0" o un "1" en función de si se va a escribir o a leer (w). Para ello, como se puede ver en el código siguiente, primero escribe en el registro TWDR (A.A.4) la dirección del esclavo junto con el bit de escritura/lectura.

```
/*************************************************************
/                                                           **
/      Función para enviar la dirección del esclavo y si    **
/      se va a escribir o leer.                             **
/                                                           **
/      PARAMETROS:                                          **
/      - ADDR -> Dirección del esclavo con el que nos vamos **
/                comunicar.                                 **
/      - w    -> Bit de escritura (0) o lectura (1).        **
/                                                           **
*************************************************************/
void send_address_i2c (uint8_t ADDR, uint8_t w) {
```

```
        TWDR = (ADDR<<1) | w; //Carga la dirección mas el bit de
escritura o lectura.
        TWCR = (1 << TWINT) | (1<<TWEN); //Limpia el bit TWINT
para comenzar la transmisión de la dirección.

        while(!(TWCR & (1 << TWINT))); //Espera a que TWINT = 0
}
```

- **Void stop().**
  Esta función se encarga de enviar una condición de STOP. Para ello, escribe un "1" en el bit TWSTO (bit 4) del registro TWCR (A.A.2).

```
/*************************************************************
/                                                            **
/      Función para enviar la condición de parada de la     **
/      comunicación.                                         **
/                                                            **
*************************************************************/
void stop_i2c() {
        TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
}
```

- **Void send_i2c(uint8_t data).**
  Esta función es la encargada de enviar el dato al esclavo indicado anteriormente. Para ello carga el byte con el dato (data) en el registro TWDR (A.A.4) y nuevamente limpia el bit TWINT de TWCR para comenzar el envío y espera hasta que este vuelve a "0".

```
 /*************************************************************
 /                                                           **
 /     Función para enviar el dato a la dirección enviada    **
 /     antes con la función send_address_i2c                 **
 /                                                           **
 /     PARAMETROS:                                           **
 /     - data -> Dato que vamos a enviar al esclavo          **
 /                                                           **
 *************************************************************/
 void send_i2c(uint8_t data) {
       // Envía el dato a la dirección previa
       TWDR = data; //Carga el dato en el registro TWDR.
       TWCR = (1 << TWINT) | (1<<TWEN); //Limpia TWINT en TWCR
   para comenzar la transmisión del dato.
       while(!(TWCR & (1 << TWINT)));
   }
```

- **Uint8_t read_i2c().**
  Esta función se encarga de leer el dato recibido de la dirección anterior. Para ello espera a que se haya enviado por completo el dato poniendo TWINT a "1", y espera que vuelva a "0" para obtener el dato.

```
/*************************************************************
/                                                           **
/      Función para leer el dato recibido de la dirección   **
/      enviada antes con la función send_address_i2c.       **
/                                                           **
*************************************************************/
uint8_t read_i2c() {
        TWCR = (1 << TWINT) | (1 << TWEN);
```

```
        while(!(TWCR & (1 << TWINT)));
        return TWDR;
}
```

4.3.2 Desarrollo de librería USART – Xbee.

Tal y como se ha comentado a lo largo de este documento, los datos de salida proporcionados por el ordenador encargado de procesar las imágenes, tienen que ser enviados de forma inalámbrica al módulo Xbee, el cual estará conectado mediante un puerto serie al microprocesador. Este puerto serie se implementará mediante el módulo USART disponible en el microprocesador [28].

Para el uso de este módulo, al igual que ocurría con el TWI, es necesario conocer los registros que controlaran dicho modulo. Para ello ver el Anexo A.B.

4.3.2.1 Implementación de la librería USART.

Una vez se han descrito todos los registros de control del módulo USART y cuáles son sus funciones, se puede continuar con el desarrollo de la librería para poder usar dicho protocolo en el microprocesador. A continuación, se va a analizar función a función la librería desarrollada para tal fin:

- **Void uart_init(uint32_t baud).**
  Esta función se encarga iniciar el bus uart a la velocidad indicada por parámetro en baudios (*baud*). Además, seguidamente se habilita el bit U2X1, por lo que se dobla la tasa de transferencia para una comunicación asíncrona, se habilita el receptor y el transmisor (poniendo a "1" los bits RXEN y TXEN), y se establece una longitud de 8 bits para los frames (poniendo a "1" los bits UCSZ11 y UCSZ10). Por último, se inicializan los buffers virtuales (son variables creadas en el software para ayudar que no se corresponden con el buffer del USART), tanto de transmisión como de recepción, a 0.

```
/************************************************************
/                                                           **
/     Función para iniciar el bus uart a la velocidad        **
/     indicada por el parametro de entrada baud, en baudios  **
/                                                           **
************************************************************/
void uart_init(uint32_t baud)
{
        UBRR1 = (F_CPU / 4 / baud - 1) / 2;
        UCSR1A = (1<<U2X1);
        UCSR1B = (1<<RXEN1) | (1<<TXEN1) | (1<<RXCIE1);
        UCSR1C = (1<<UCSZ11) | (1<<UCSZ10);
        tx_buffer_head = tx_buffer_tail = 0;
        rx_buffer_head = rx_buffer_tail = 0;
}
```

- **Void uart_putchar(uint8_t c).**
  Esta función es la encargada de escribir un Byte en el bus. Para ello, espera a que haya espacio suficiente en el buffer virtual para introducir el Byte, y, cuando es así lo introduce. Además, vuelve a habilitar el receptor y el transmisor, y el uso de interrupciones en el receptor y en la bandera UDR (escribir a "1" los bits RXEN y

TXEN para habilitar transmisor y receptor, y los bits RXCIE y UDRIE para las interrupciones en el receptor y en la bandera UDR).

```
/*************************************************************
/                                                            **
/      Función para escribir un byte en el bus uart.         **
/                                                            **
*************************************************************/
void uart_putchar(uint8_t c)
{
       uint8_t i;

       i = tx_buffer_head + 1;
       if (i >= TX_BUFFER_SIZE) i = 0;
       while (tx_buffer_tail == i); //Espera hasta que haya
                                          espacio en el buffer.
       tx_buffer[i] = c;
       tx_buffer_head = i;
       UCSR1B = (1<<RXEN1) | (1<<TXEN1) | (1<<RXCIE1) |
      (1<<UDRIE1);
}
```

- **Interrupción para transmitir.**
  Este fragmento de código, es una interrupción que se ejecuta cuando el buffer esta vacío y puede transmitir más bytes, lo cual ocurre cuando el bit UDRE está a "1", y por lo tanto se puede transmitir información (grabo los datos en el registro UDR1).

```
/*********************************************************
/                                                       **
/      Interrupción para Transmitir                      **
/                                                       **
*********************************************************/
ISR(USART1_UDRE_vect)
{
       uint8_t i;

       if (tx_buffer_head == tx_buffer_tail) {
              UCSR1B = (1<<RXEN1) | (1<<TXEN1) | (1<<RXCIE1);
       }
       else {
              i = tx_buffer_tail + 1;
              if (i >= TX_BUFFER_SIZE) i = 0;
              UDR1 = tx_buffer[i];
              tx_buffer_tail = i;


       }
}
```

- **Uint8_t uart_getchart().**
  Esta función se encarga de obtener el byte que está guardado en el buffer, el cual se guarda ahí al ejecutarse la interrupción producida cuando hay datos en el buffer sin leer (RXCn a "1").

```
/*********************************************************
/                                                       **
/  Función para leer un byte del buffer                 **
/                                                       **
*********************************************************/
uint8_t uart_getchar(void)
```

```
    {
        uint8_t c, i;

        while (rx_buffer_head == rx_buffer_tail);

        i = rx_buffer_tail + 1;
        if (i >= RX_BUFFER_SIZE) i = 0;
        c = rx_buffer[i];
        rx_buffer_tail = i;
        return c;
    }
```

- **Interrupción para recibir.**
  Este fragmento de código es una interrupción que se ejecuta cuando el buffer tiene datos que aún no han sido leídos (RXCn a "1"). Cuando se produce esta interrupción, se guardan los datos disponibles en el registro UDR, y se copian al buffer virtual de recepción.

```
/*************************************************************
/                                                           **
/      Interrupción para Recibir                             **
/                                                           **
*************************************************************/
ISR(USART1_RX_vect)
{
        uint8_t c, i;

        c = UDR1;
        i = rx_buffer_head + 1;
        if (i >= RX_BUFFER_SIZE) i = 0;
        if (i != rx_buffer_tail) {
                rx_buffer[i] = c;
                rx_buffer_head = i;
        }
}
```

- **Uint8_t uart_available().**
  Esta función devuelve el número de bytes disponibles en el buffer.

```
/*************************************************************
/                                                           **
/      Función que devuelve el número de bytes disponibles   **
/      en el buffer. Llamar a esta función antes de leer un **
/      byte para esperar a que haya un byte en el buffer.    **
/                                                           **
*************************************************************/
uint8_t uart_available(void)
{
        uint8_t head, tail;

        head = rx_buffer_head;
        tail = rx_buffer_tail;
        if (head >= tail) return head - tail;
        return RX_BUFFER_SIZE + head - tail;
}
```

- **Void uart_print_P(const char *str).**
  Esta función es una función auxiliar creada para hacer el envío de cadenas de caracteres, y así no estar limitados a hacer envíos Byte a Byte. Para ello únicamente

fragmenta la cadena de caracteres de entrada en bytes y los va enviando uno a uno con la función *uart_putchar()*.

```
/*************************************************************
/                                                           **
/      Función para escribir una cadena de caracteres en el  **
/      buffer.                                              **
/                                                           **
*************************************************************/
void uart_print_P(const char *str)
{
        char c;
        while (1) {
                c = pgm_read_byte(str++);
                if (!c) break;
                uart_putchar(c);
        }
}
```

4.3.3 Desarrollo librería de control del driver PCA9685 (Driver I2C).

Por último, en esta primera fase de desarrollo de librerías, fue necesario, con el fin de simplificar el código principal para el control del prototipo, implementar una librería encargada de llevar a cabo la configuración y la comunicación con el driver PCA9685. Como bien se ha ido diciendo a lo largo del documento, la comunicación con este driver se realiza a través del bus I2C (en este caso TWI), por lo que la librería desarrollada anteriormente (4.3) facilita considerablemente el desarrollo de esta nueva librería.

4.3.3.1 Registros de control del driver PCA9685.

Tal y como se puede ver en el Anexo A.C este dispositivo tiene una gran lista de registros de control, para activar cada una de sus funcionalidades [30]. Por ello, únicamente se detallarán los registros más importantes, ya que el resto de registros, únicamente se utilizan para encender o apagar la salida N.

Todos estos registros disponibles, no son más que direcciones o comandos que hay que enviar en el bus TWI al dispositivo PCA9685, con el fin de ejecutar la tarea correspondiente.

Por ejemplo, en la siguiente figura (Figura 4-26), puede verse un ejemplo típico de trama de escritura TWI enviada a este driver, donde en primer lugar, seguido de una condición de START, se envía la dirección Hardware del dispositivo y el bit de escritura/lectura. Si esta primera parte se recibe correctamente, el esclavo contestará con un ACK, y seguidamente se enviará el registro de control correspondiente para la tarea que se quiera realizar. Nuevamente, si se ha recibido correctamente, el esclavo envía otro ACK. Para finalizar, se envían los datos necesarios para configurar el registro de control anterior.

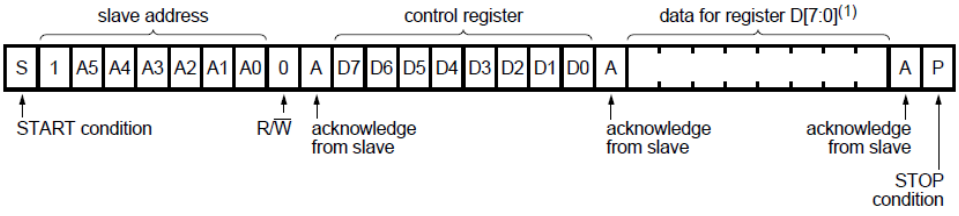


Figura 4-26. Trama TWI típica para la escritura de datos en el PCA9685.

- **Registro de Control MODE1.**

De todos los registros de control disponibles en el driver, existe un registro al cual se denomina MODE1, al cual se le llama con el comando 0x00. Este registro se divide en 8 bits con diferentes funciones (Tabla 2). Este registro, es el encargado de configurar el driver de dos modos diferentes, o bien usando un reloj externo, o el propio reloj interno. Además, también puede ponerse a "Dormir" mientras está a la espera de instrucciones. Este registro es muy importante de configurar adecuadamente para la función que se quiera desempeñar, ya que, si no es así, el dispositivo no funcionará.

Tabla 2. Registro MODE1.

Bit	Símbolo	Acceso	Valor	Descripción
7	RESTART	R/W	0	Deshabilita el reinicio del dispositivo.
			1	Habilita el reinicio del dispositivo.
6	EXTCLK	R/W	0	Usa el reloj interno.
			1	Usa el reloj externo conectado al pin EXTCLK.
5	AI	R/W	0	Deshabilita el autoincremento del registro.
			1	Habilita el autoincremento del registro.
4	SLEEP	R/W	0	Modo normal.
			1	Modo de baja potencia. Deshabilita el oscilador.
3	SUB1	R/W	0	No responde a la Subdirección 1.
			1	Responde a la Subdirección 1.
2	SUB2	R/W	0	No responde a la Subdirección 2.
			1	Responde a la Subdirección 2.
1	SUB3	R/W	0	No responde a la Subdirección 3.
			1	Responde a la Subdirección 3.
0	ALLCALL	R/W	0	No responde a la dirección *LED All Call.*
			1	Responde a la dirección *LED All Call.*

- **Registro de control PRE_SCALE.**

Otro registro que puede resultar muy útil en determinadas aplicaciones, es el registro PRE_SCALE, el cual se corresponde con el comando 0xFE. Este registro se encarga de definir la frecuencia de modulación del PWM de la señal de salida. Por defecto, el hardware fuerza que el mínimo valor que puede ser cargado en este registro es "3".

El valor de este registro se determina con la siguiente formula:

$$prescale = round\left(\frac{osc\_clock}{4096\ x\ update\_rate}\right) - 1,$$

Donde el valor *update_rate* es la frecuencia de modulación de la señal de salida deseada. Por ejemplo, para una frecuencia de modulación de 200Hz y la frecuencia del reloj de 25MHz:

$$prescale = round\left(\frac{25MHz}{4096 \times 200}\right) - 1 = 30\ (0x1E)$$

Un requisito indispensable para poder usar este registro, es que el bit SLEEP del registro MODE1 (0) este establecido a “1”.

- **Registro de control LEDn_ON_L/H y LEDn_OFF_L/H.**

Por último, otros registros importantes, son los encargados de encender y apagar cada una de las salidas, y con qué PWM. Para ello se utilizan 4 registros para cada salida (en total son 64 registros diferentes), que son: LEDn_ON_L, LEDn_ON_H, LEDn_OFF_L y LEDn_OFF_H, donde *n* indica a que salida corresponde el registro (de 0 a 15).

En los registros LEDn_ON_L/H, se indica donde se quiere que tenga lugar el flanco de subida de la señal PWM, mientras que en los registros LEDn_OFF_L/H se indica donde se quiere que tenga lugar el flanco de bajada de la señal PWM.

Por ejemplo, si se quiere que la salida 0 tenga un *Delay* (tiempo que tarda en haber un flanco de subida) del 10% y *Duty Cicle* (tiempo que la señal se mantiene en alto) del 20%, los registros se configurarían de la siguiente forma (hay que tener en cuenta que un ciclo va de 0 a 4095):

$$Delay\ time = 10\% = 409.6\ \approx 410\ counts = 19Ah$$

$$LED0\_ON\_H = 1h; LED\_0\_ON\_L = 9Ah$$

$$Duty\ Cicle = 20\% = 819.2 \approx 819\ counts$$

$$LED\ off = 410 + 819 = 1228 = 4CCh$$

$$LED0\_OFF\_H = 4h; LED0\_OFF\_L = CCh$$

Como se puede observar, tanto en los dos registros de encendido como en los de apagado, la H indica que se escribirá en él, 4 bits más significativos, y la L indica que se escribirá el byte menos significativo de los valores de cuenta correspondientes a cada tiempo. Por lo tanto, si se hiciese esa configuración en el chip PCA9685, se obtendría una salida como la que se muestra a continuación (Figura 4-27):

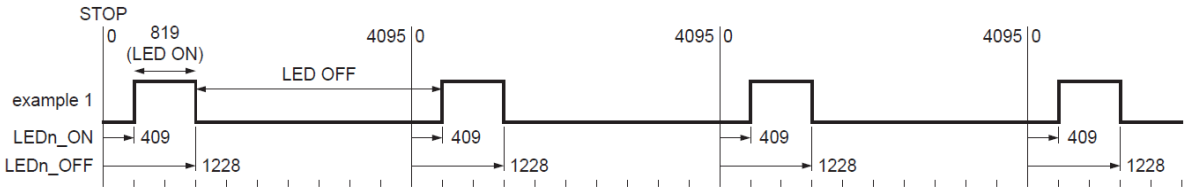


Figura 4-27: Ejemplo una salida del chip PCA9685 configurada con 10% de delay y un 20% de duty cicle.

4.3.3.2 Implementación de la librería PCA9685.

Una vez se han descrito los registros de control más importantes del chip PCA9685 y cuáles son sus funciones, se puede continuar con el desarrollo de la librería para poder configurar de la manera deseada dicho chip. A continuación, se va a analizar función a función la librería desarrollada para tal fin:

- **Void begin(uint8_t _i2caddr).**
  Esta función, como bien indica el nombre, es la encarga de iniciar el driver PCA9685. Para ello, en primer lugar, inicia el protocolo TWI (o I2C) en el microprocesador, para continuar llamando a la función *reset*, que se explicara más adelante. Termina llamando a la función *setPWMFreq*, con la que se asignará una frecuencia por defecto a la modulación del PWM.

```
/************************************************************
/                                                          **
/      Función para iniciar el driver PCA9685, en la cual   **
/      primero inicia el i2c del micro y se resetea y       **
/      establece la frecuencia en la que va a funcionar el  **
/      driver.                                              **
/                                                          **
************************************************************/
void begin(uint8_t _i2caddr){
       init_i2c();
       reset(_i2caddr);
       setPWMFreq(_i2caddr, 200);
}
```

- **Void reset(uint8_t _i2caddr).**
  Esta función es la encargada de resetear el driver. Para ello, llama a la función *write_byte*, la cual envía un byte a una dirección determinada a través del bus TWI. En este caso, se envía a la dirección del registro MODE1 (0) el valor 0x80, de forma que el bit RESTAR se pone a "1". Para terminar, espera 10ms para que dé tiempo a ejecutarse el Reset.

```
/************************************************************
/                                                          **
/      Función para resetear el driver.                     **
/      Se escribe en el registro MODE1 el valor 0x80        **
/                                                          **
************************************************************/
void reset(uint8_t _i2caddr) {
       write_byte(_i2caddr, PCA9685_MODE1, 0x80);
       _delay_ms(10);
}
```

- **Void setPWMFreq(uint8_t _i2caddr, float freq).**
  Esta función es la más compleja de todas, ya que es en esta donde se configura el driver. Para ello, en primer lugar, se calcula el valor del registro PRE_SCALE a partir de la frecuencia de modulación del PWM deseada (parámetro de entrada freq). Después, para poder configurar el registro PRE_SCALE, tal y como se ha visto en 0, es necesario poner el dispositivo en SLEEP, para ello hay que poner a "1" el bit 4 del registro MODE1 (0). Una vez configurado el PRE_SCALE, el registro MODE1, se tiene que retornar a su valor anterior. Por último, pasados 5ms para que dé tiempo al micro a procesar los comandos, se activa el autoincremento en el registro MODE1.

```
/************************************************************
/                                                          **
/      Función para establecer la frecuencia de             **
/      funcionamiento del driver mediante el registro       **
/      PRESCALE.                                            **
/                                                          **
************************************************************/
```

```
void setPWMFreq(uint8_t _i2caddr, float freq) {

	freq *= 0.9;  //Factor de corrección de la frecuencia.
	float prescaleval = 25000000;
	prescaleval /= 4096;
	prescaleval /= freq;
	prescaleval -= 1;

	uint8_t prescale = floor(prescaleval + 0.5);

	uint8_t oldmode = read_byte(_i2caddr, PCA9685_MODE1);
	uint8_t newmode = (oldmode & 0x7F) | 0x10;
	write_byte(_i2caddr, PCA9685_MODE1, newmode);
	write_byte(_i2caddr, PCA9685_PRESCALE, prescale);
	write_byte(_i2caddr, PCA9685_MODE1, oldmode);
	_delay_ms(5);
	write_byte(_i2caddr, PCA9685_MODE1, oldmode | 0xa0);
}
```

- **Void setPWM(uint8_t _i2caddr, uint8_t pin, uint16_t on, uint16_t off).**
  Esta función es la encargada de configurar para un puerto de salida determinado (parámetro de entrada pin), un valor de PWM determinado, tal y como se ha explicado en □. Para ello, se inicia el bus TWI enviando una condición de START, seguidamente se envía la dirección del esclavo del cual se quiere activar la salida, seguido de un "0" (indica que se va a escribir en el esclavo). A continuación, se manda la dirección del registro de control correspondiente a la salida que se quiere activar. Como anteriormente se ha activado la función de autoincremento, se pueden escribir seguidamente los valores de los 4 registros LEDn_ON/OFF_L/H (esto se puede hacer ya que las direcciones de los cuatro registros son consecutivas, sea cual sea la salida a activar). Por último, se envía una condición de STOP.

```
/************************************************************
/                                                          **
/      Función para establecer el valor de pwm de una salida **
/      del driver, donde on es el momento de subida del      **
/      flanco, y off el momento de bajada del flanco.        **
/                                                          **
************************************************************/
void setPWM(uint8_t _i2caddr, uint8_t pin, uint16_t on, uint16_t
            off) {
	start_i2c();
	send_address_i2c(_i2caddr, 0);
	send_i2c(LED0_ON_L+4*pin);
	send_i2c(on);
	send_i2c(on>>8);
	send_i2c(off);
	send_i2c(off>>8);
	stop_i2c();
}
```

- **Void write_byte(uint8_t _i2caddr, uint8_t addr, uint8_t d).**
  Esta función es muy similar a la anterior, con la única diferencia que esta función únicamente escribe un byte (parámetro de entrada d) en un esclavo determinado (parámetro _i2caddr) y en un registro de control determinado (parámetro addr).

```
/************************************************************
/                                                          **
/      Función para escribir un byte d en una direccion      **
```

```
/      del driver addr.                                              **
/                                                                    **
**********************************************************************/
void write_byte(uint8_t _i2caddr, uint8_t addr, uint8_t d) {
       start_i2c();
       send_address_i2c(_i2caddr, 0);
       send_i2c(addr);
       send_i2c(d);
       stop_i2c();
}
```

- **uint8_t read_byte(uint8_t _i2caddr, uint8_t addr).**
  Por último, se encarga de leer un registro de control (parámetro addr) en un esclavo determinado (parámetro _i2caddr), de forma que devuelva el valor leído en el registro.

```
/**********************************************************
/                                                         **
/      Función para leer un byte en una direccion        **
/      del driver addr.                                   **
/                                                         **
**********************************************************/
uint8_t read_byte(uint8_t _i2caddr, uint8_t addr) {
       start_i2c();
       send_address_i2c(_i2caddr, 0);
       send_i2c(addr);
       return read_i2c();
}
```

4.4 Configuración y Pruebas.

4.4.1 Configuración y Test de funcionamiento de la comunicación inalámbrica.

En esta nueva fase, el objetivo fundamental consistió en comprobar el funcionamiento de la librería creada para el control del puerto serie con la placa Teensy 2.0 (4.3.2), la cual, como bien se ha explicado, se usará para recibir y procesar la información recibida por el dispositivo Xbee. Además, también se comprobará en esta fase si el diseño del segundo PCB (4.2.1) se ha realizado correctamente y las conexiones entre el dispositivo Xbee y el procesador funcionan.

Para ello, en primer lugar, se conectaron el dispositivo Xbee y la placa Teensy 2.0 en los zócalos creados en el PCB desarrollado, para posteriormente cargar un código de prueba en el microprocesador, de forma que se probasen todas las funciones creadas en la librería UART y conseguir establecer una comunicación inalámbrica con el dispositivo Xbee.

El montaje del esquema realizado fue el siguiente:

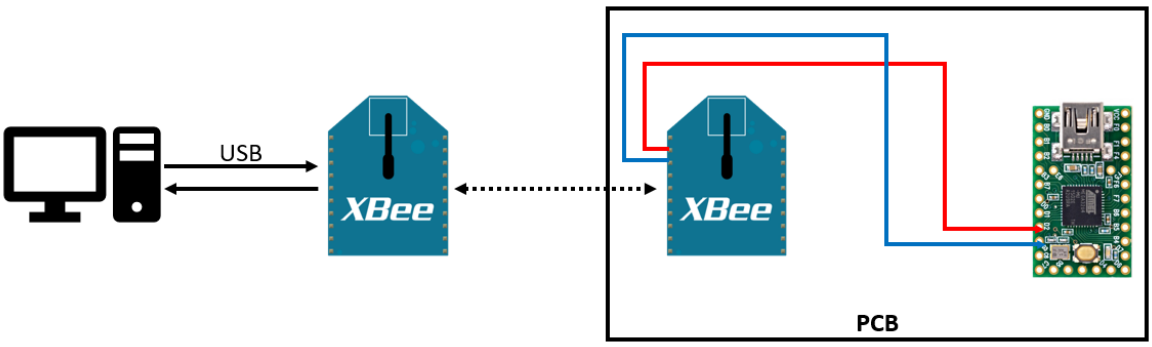


Figura 4-28: Esquema de conexión para test de comunicación inalámbrica.

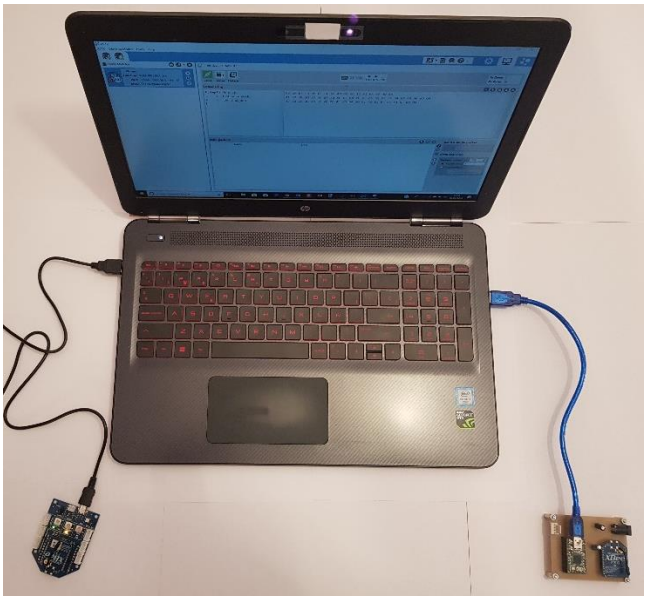


Figura 4-29: Montaje Real.

Una vez conectado todo, se ha cargado un código de prueba en el microprocesador, tal y como se indica en 3.2.2.2, de forma que se probasen todas las funciones creadas en la librería encargada de hacer funcionar el módulo USART del microprocesador, y así, posteriormente asegurar un correcto funcionamiento del prototipo a desarrollar.

El código creado para tal fin ha sido el siguiente:

```
int main(void)
{
	uint8_t c = 0xAB;
	// CONFIGURACIÓN INICIAL.
	uart_init(9600); //Inicio el bus UART a 9600 baudios
	                    por segundo
	LED_CONFIG;          //Configuro el LED disponible en la
	                      placa Teensy 2.0

	uart_print("Ejemplo Mensaje\r\n"); //Envío un mensaje
	                                     usando el Xbee.
	while (1) {
		if (uart_available()) { //Espero a que haya al
		                         menos un byte en el buffer.
			c = uart_getchar(); //Obtengo el byte
			                     disponible en el buffer.

			if (c == 0x41){ //Si he recibido un 0x41
			                 enciendo el LED de la placa
				LED_ON;
				uart_print("  -->  Led
				            encendido\r\n");
			}
			else{               //Sino, la apago
				LED_OFF;
				uart_print("  -->  Led apagado\r\n");
			}
		}
	}
}
```

Tal y como se puede observar, en primer lugar, se hace una configuración inicial del dispositivo USART, el cual se inicia a una tasa de 9600 baudios por segundo, y seguidamente se indica a la placa Teensy 2.0 que se va a hacer uso del LED integrado en ella.

Una vez se ha realizado esta configuración, se envía un primer mensaje para comprobar que se recibe correctamente en el otro dispositivo conectado al ordenador y que, por lo tanto, la transmisión de información funciona.

Seguidamente, se espera hasta que haya algún byte disponible en el buffer de recepción, el cual será enviado mediante el dispositivo Xbee conectado al ordenador haciendo uso del software XCTU. Una vez este el byte enviado en el buffer de recepción, se obtendrá y se comprobará su valor, de forma que si el valor corresponde con un 0x41 (se corresponde con la letra “A” en ASCII), se encenderá el Led de la placa Teensy 2.0, y se apagará cuando se obtenga un valor diferente (Figura 4-30).

De esta forma, si todo ha funcionado como se esperaba, con este pequeño Script, se han comprobado si las conexiones realizadas en el PCB correspondientes a la unión del Xbee con el microprocesador funcionan, y lo que es más importante, si la librería implementada

para el uso del módulo USART del microprocesador funciona correctamente. Para ver con más detalle que realizan cada una de las funciones usadas en este script, ir a la sección 4.3.2.

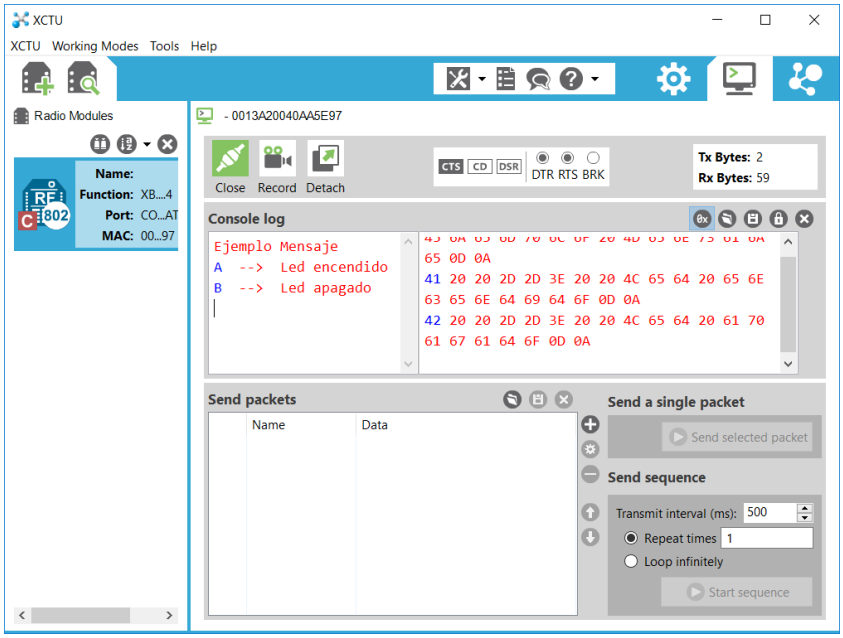


Figura 4-30: Resultado de ejecución en XCTU.

4.4.2 Configuración y Test de funcionamiento del bus y driver I2C.

En esta segunda fase de comprobación del funcionamiento, se probarán si las conexiones entre el microprocesador y el driver I2C se han realizado correctamente. Además, en esta fase, también se comprobará el correcto funcionamiento de la librería I2C, o TWI como se ha llamado a lo largo de este documento, y la librería encargada de configurar el driver PCA9685.

Para ello, se conectarán las dos placas PCBs desarrolladas mediante un cable que llevará 4 señales diferentes, SDA, SCL, Vcc y GND. A su vez, al PCB con el driver I2C, se conectará un led a la salida 0, de forma que se pueda controlar su encendido y apagado mediante el software grabado en el microprocesador. En esta prueba, no se utilizará el dispositivo Xbee para nada, por lo que puede desconectarse.

El montaje realizado fue el siguiente:

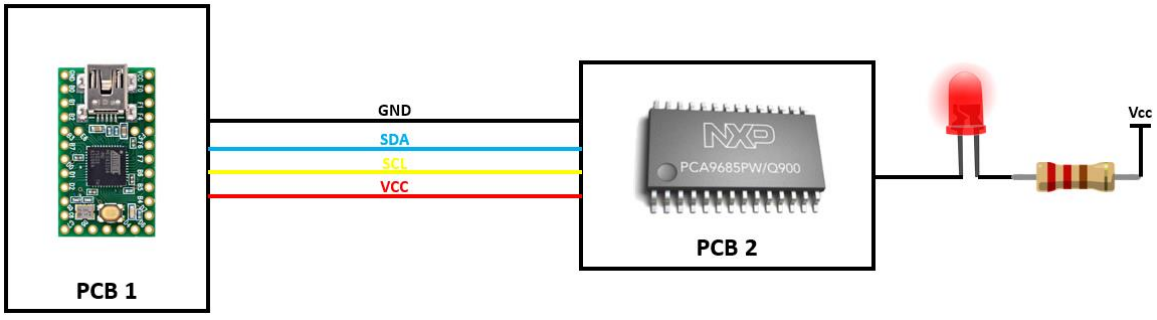


Figura 4-31: Esquema de conexión para test del Driver PCA9685.

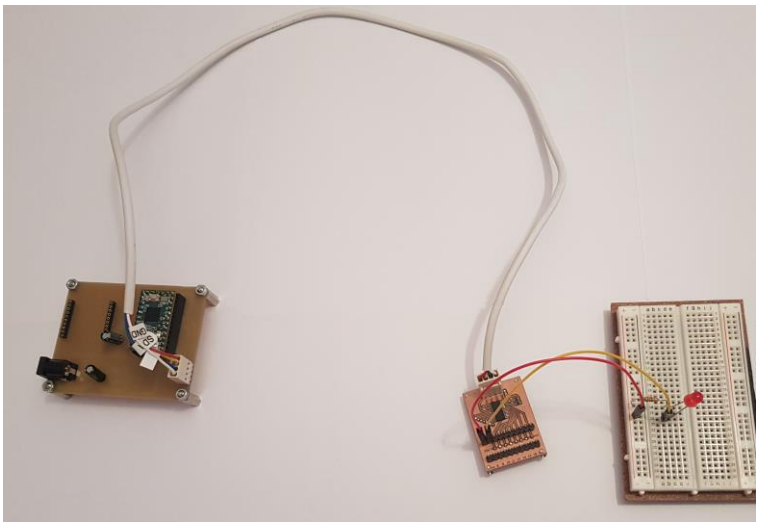


Figura 4-32: Montaje Real.

Una vez realizadas todas las conexiones, se desarrolló un script que hiciese uso de las librerías I2C y PCA9685 (para más información ver 4.3 y 4.3.3). En este script, el único objetivo era hacer parpadear el Led conectado a la salida 0 del driver con un PWM del 50%.

El script desarrollado fue el siguiente:

```
#include <avr/io.h>
#include "i2c.h"
#include <util/delay.h>
#include "PCA9685.h"
#include <stdio.h>
#include <string.h>

int main(void)
{

	//INICIALIZACIÓN DE DIRECCIONES DE LOS ESCLAVOS.
	uint8_t addrMaster1 = 0x4D;
	int pin = 0;

	//INICIALIZACIÓN DEL DRIVER I2C:
	begin(addrMaster1);
	setPWMFreq(addrMaster1, 1600);

	//APAGO LA SALIDA 0.
	setPWM(addrMaster1, pin, 4096, 0);

	while(1){

		//ENCIENDO LED.
		setPWM(addrMaster1, pin, 0, 2048);
		_delay_ms(500);

		//APAGO LED.
		setPWM(addrMaster1, pin, 4096, 0);
		_delay_ms(500);
	}
}
```

Como se puede observar, en este script, en primer lugar, se inicia el bus I2C, y se establece que la comunicación va a ser con el esclavo con dirección 0x4D con una frecuencia de PWM de 1600 Hz. Seguidamente se apaga la salida arrancar desde el estado apagado. Por último, en bucle, se enciende y se apaga el led conectado al pin 0 con un PWM del 50% (2048 valores de 4096) durante 500 ms, dando así lugar a un parpadeo.

4.4.3 Configuración y Test del sistema completo.

En esta fase, el objetivo principal era combinar los resultados de ambas fases anteriores, obteniendo así una primera versión del prototipo al completo.

Para ello, se juntaron ambos esquemas de conexión en uno solo de la siguiente forma:

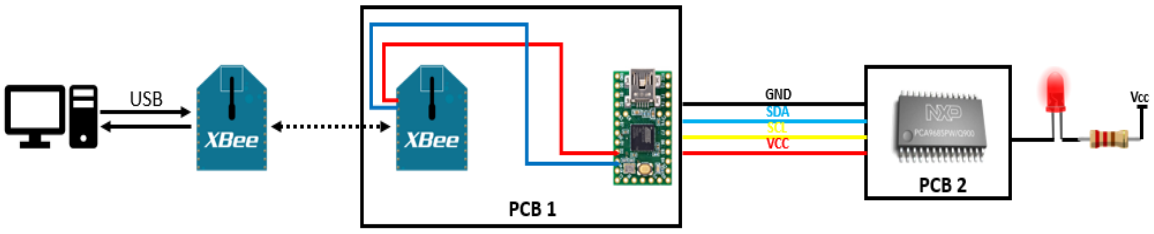


Figura 4-33: Esquema de conexión completo.

Para probar su funcionamiento, se desarrolló un script con el objetivo de enviar un valor desde el ordenador mediante el Xbee, el cual sería recibido por el otro módulo Xbee montado en el PCB1. Este, enviará el valor recibido al microprocesador a través del puerto serie. Una vez se encuentre el valor en el microprocesador, el script implementado enviará una configuración al driver PCA9685 con un valor de PWM determinado en función del valor que se haya recibido (por ejemplo, si recibo un 0 apago el led, si recibo un 1 lo enciendo al 10%, etc.) obteniendo una intensidad de luz determinada en el Led.

Por lo tanto, el script desarrollado es el siguiente:

```
#include <avr/io.h>
#include "i2c.h"
#include "Xbee.h"
#include <util/delay.h>
#include "PCA9685.h"

int main(void)
{
	//INICIALIZACIÓN DEL DRIVER I2C:
	 begin();
	 setPWMFreq(1600);

	 //INICIALIZACIÓN DEL XBEE:
	 uint8_t c = 0xAB;
	 int i = 0;

	 CPU_PRESCALE(0);
	 uart_init(BAUD_RATE);
	 LED_CONFIG;
```

```
        setPWM(0, 4096, 0);

        //BUCLE
        while(1){
               if (uart_available()) {
                     c = uart_getchar();

                     if (c == 0x30){ //Valor 0
                            LED_OFF;
                            setPWM(0, 4096, 0);
                     }
                     else if (c == 0x31){ //Valor 1
                            LED_ON;
                            setPWM(0, 0, 4000);
                     }
                     else if (c == 0x32){ //Valor 2
                            LED_ON;
                            setPWM(0, 0, 3500);
                     }
                     else if (c == 0x33){ //Valor 3
                            LED_ON;
                            setPWM(0, 0, 3000);
                     }
                     else if (c == 0x34){ //Valor 4
                            LED_ON;
                            setPWM(0, 0, 2500);
                     }
                     else if (c == 0x35){ //Valor 5
                            LED_ON;
                            setPWM(0, 0, 2000);
                     }
                     else{ //Cualquier valor
                            LED_ON;
                            setPWM(0, 0, 1500);
                     }
               }
        }
}
```

Tal y como se puede observar en el anterior fragmento de código, en esta ocasión se usarán todas las librerías desarrolladas. Al igual que en los códigos anteriores, en un primer lugar se inicializan tanto el driver PCA9685 como el módulo USART conectado al Xbee. Seguidamente, en un bucle, se obtiene el valor enviado con el ordenador al Xbee, y en función del valor obtenido (0, 1, 2, 3, 4 o 5), se envía un valor de PWM determinado a la salida 0 del PCA9685, de forma que el Led conectado a ella se encienda con diferentes intensidades.

4.5 Software de Control Final

Una vez se ha comprobado el funcionamiento del sistema al completo con una versión del software sencilla, se procederá a desarrollar una versión más completa y definitiva.

Para ello, es importante tener en cuenta, que hasta ahora solamente se ha usado un esclavo (PCA9685) del cual únicamente se ha activado una salida. Esta versión en cambio, tiene que ser modular, es decir que puedan cambiarse con facilidad la cantidad de esclavos, y por lo tanto la cantidad de salidas a activar, ya que una de las especificaciones del prototipo, es que tiene que controlar hasta 128 motores de manera independiente. Por ello, es necesario que el ordenador envíe los datos al microprocesador de forma masiva, y que posteriormente el microprocesador los procese y active los diferentes motores. Por lo tanto, para poder hacer esto, es necesario definir un formato de envío de datos, de manera que luego se puedan procesar de forma sencilla y sin gastar mucho tiempo de proceso.

4.5.1 Formato de datos enviados a través de Xbee.

Los datos que se van a enviar desde el ordenador, indicaran que salida de las 128 posibles se activará y con qué potencia o modulación PWM lo hará. Por ello, una de las formas más sencillas de hacer este envío, es mediante una matriz o vector, donde la posición del dato en esta matriz o vector indica de que salida se trata, es decir, la posición 0 del vector indica que se activará la salida 0 del esclavo 1, la posición 40 indica que se trata de la salida 8 del esclavo 3, y el valor que hay en esa posición indica el PWM de la señal que se enviará a dicha salida.

Además, el valor que se envía, debe ser un valor del porcentaje de la señal PWM, de forma que, si se tiene en cuenta que el dispositivo Xbee envía al microprocesador los datos en formato hexadecimal, de forma que, si se calcula el valor en decimal, se obtiene el valor en porcentaje (si el valor es 0x32 se corresponde con un 50%).

De esta forma, la matriz de datos enviada, tendrá el mismo tamaño que la cantidad de salidas a activar, por lo que, si se tienen 128 motores, se enviará una matriz de 128 valores (Figura 4-34).

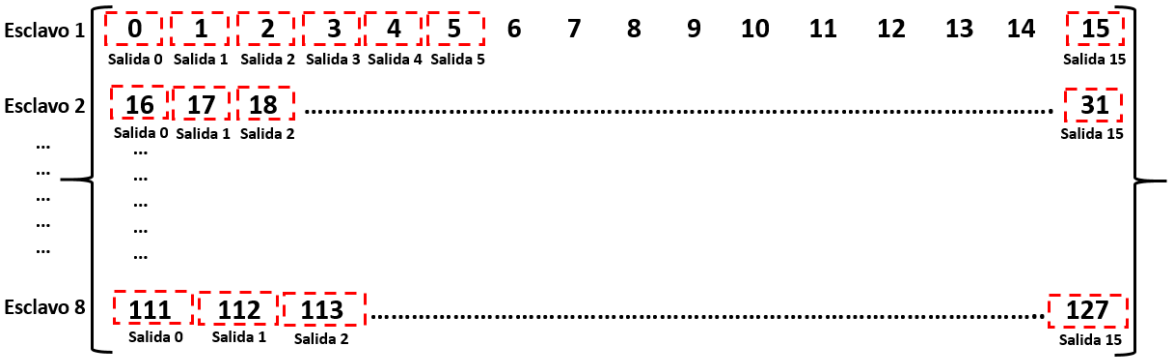


Figura 4-34: Matriz de datos enviada.

Por lo tanto, teniendo en cuenta el formato de los datos, si en las posiciones 0 de la matriz, se recibe un 0x32, significaría, por lo tanto, que la salida 0 del esclavo 1 se configurará con una señal PWM del 50%. Ya que el driver PCA9685 proporciona salidas PWM con una

resolución de 12 bits, la señal PWM puede ir de 0 a 4095 valores, de forma que una señal del 50%, significaría tener una señal donde estará a "1" durante 2047 valores y a "0" durante el resto (ver 4.3.1.1 para más información sobre la configuración de la salida PWM).

Una vez determinado el formato de envío y recepción de los datos, se continuará con la implementación del script software encargado del procesamiento de los datos y funcionamiento del prototipo.

4.5.2 Implementación del Software de control del prototipo.

Para poder implementar el software de control del prototipo, es necesario tener muy claro cómo va a ser el funcionamiento de este.

El software desarrollado está formado por una función principal, la cual puede verse a continuación:

```
int main(void)
{
	//ENCENDIDO DEL LED DEL MICRO.
	LED_CONFIG;
	LED_ON;

	//INICIALIZACIÓN DE DIRECCIONES DE LOS ESCLAVOS.
	uint8_t addrMaster1 = 0x4D;
	uint8_t addrMaster2 = 0x4D;
	uint8_t addrMaster3 = 0x4D;
	uint8_t addrMaster4 = 0x4D;
	uint8_t addrMaster5 = 0x4D;
	uint8_t addrMaster6 = 0x4D;
	uint8_t addrMaster7 = 0x4D;
	uint8_t addrMaster8 = 0x4D;

	//INICIALIZACIÓN DEL DRIVER I2C: (CAMBIAR CUANDO AÑADAMOS
MAS ESCLAVOS)
	begin(addrMaster1);
	setPWMFreq(addrMaster1, 1600);
	begin(addrMaster2);
	setPWMFreq(addrMaster2, 1600);
	begin(addrMaster3);
	setPWMFreq(addrMaster3, 1600);
	begin(addrMaster4);
	setPWMFreq(addrMaster4, 1600);
	begin(addrMaster5);
	setPWMFreq(addrMaster5, 1600);
	begin(addrMaster6);
	setPWMFreq(addrMaster6, 1600);
	begin(addrMaster7);
	setPWMFreq(addrMaster7, 1600);
	begin(addrMaster8);
	setPWMFreq(addrMaster8, 1600);

	//INICIALIZACION VARIABLES.
	int dimension = 16;
	uint8_t array1[dimension];
	int valor = 0;
	int pwm;

	//INICIALIZACIÓN DEL XBEE:
	CPU_PRESCALE(0);  // run at 16 MHz
```

```
        uart_init(BAUD_RATE);

        //SECUENCIA DE ENCENDIDO.
        uart_print("\r\nIniciando Dispositivo...\r\n");
        encendido(dimension, addrMaster1);
        encendido(dimension, addrMaster2);
        encendido(dimension, addrMaster3);
        encendido(dimension, addrMaster4);
        encendido(dimension, addrMaster5);
        encendido(dimension, addrMaster6);
        encendido(dimension, addrMaster7);
        encendido(dimension, addrMaster8);
        uart_print("\r\nListo para la comunicacion:\r\n");

        while(1){

                if (uart_available() >= dimension) {

                        for(int i = 0; i < dimension; i++){
                                array1[i] = uart_getchar();
                        }
                        int pin = 0;
                        for(int i=0; i < dimension; i++){
                                int a;
                                uint8_t addrM = 0x4D;

                                a = array1[i];
                                valor = 100 - a;
                                pwm = floor(valor*40.95);

                                if(pin <= 15){
                                        addrM = addrMaster1;
                                }
                                else if(pin >= 16 && pin <= 31){
                                        addrM = addrMaster2;
                                }
                                else if(pin >= 32 && pin <= 47){
                                        addrM = addrMaster3;
                                }
                                else if(pin >= 48 && pin <= 63){
                                        addrM = addrMaster4;
                                }
                                else if(pin >= 64 && pin <= 79){
                                        addrM = addrMaster5;
                                }
                                else if(pin >= 80 && pin <= 95){
                                        addrM = addrMaster6;
                                }
                                else if(pin >= 96 && pin <= 111){
                                        addrM = addrMaster7;
                                }
                                else if(pin >= 112 && pin <= 127){
                                        addrM = addrMaster8;
                                }
                                else{
                                        addrM = addrMaster1;
                                }

                                if(pwm == 0){
                                        LED_ON;
                                        setPWM(addrM, pin, 4095, 0);
                                }
                                else{
```

```
                                setPWM(addrM, pin, 0, pwm);
                        }
                        pin++;
                }
                uart_print(" Fin\r\n");
        }
    }
}
```

Tal y como se puede observar, en esta función principal se realiza en primer lugar la inicialización de variables, como pueden ser las direcciones de los esclavos a utilizar y otras variables auxiliares, además de las inicializaciones del bus I2C para cada uno de los esclavos y el módulo USART.

A continuación, se realiza una secuencia de encendido de cada uno de los esclavos, donde se encenderán y apagarás todas las salidas de cada esclavo de forma secuencial, con el objetivo de observar si hay algún problema en alguna de las salidas. Para realizar esta tarea, se ha desarrollado una función auxiliar denominada *encendido()*.

```
void encendido(int dimension, uint8_t addrMaster1){
	for(int j = 0; j < dimension; j++){
		setPWM(addrMaster1, j, 4096, 0);
	}

	for(int times = 0; times < 1; times++){
		for (int j = 0; j < dimension; j++){
			if (j==0){
				setPWM(addrMaster1, j, 0, 0);
				setPWM(addrMaster1, j+1, 4096, 0);
			}
			else{
				setPWM(addrMaster1, j-1, 4096, 0);
				setPWM(addrMaster1, j, 0, 0);
				setPWM(addrMaster1, j+1, 4096, 0);
			}
			_delay_ms(50);
		}
		for (int i = 0; i < dimension; i++){
			if (i==15){
				setPWM(addrMaster1, 15-i, 0, 0);
				setPWM(addrMaster1, 15-i+1, 4096, 0);
			}
			else{
				setPWM(addrMaster1, 15-i-1, 4096, 0);
				setPWM(addrMaster1, 15-i, 0, 0);
				setPWM(addrMaster1, 15-i+1, 4096, 0);
			}
			_delay_ms(50);
		}
	}

	for(int j = 0; j < dimension; j++){
		setPWM(addrMaster1, j, 4096, 0);
	}
}
```

A continuación, una vez se ha realizado esa secuencia de encendido, comienza la adquisición de datos. Para ello, el sistema esperará hasta haber recibido una matriz de 128 valores completa. Una vez esto ocurra, se construirá un vector de ese tamaño, donde previamente los valores obtenidos se convierten de formato hexadecimal (formato con el cual se reciben a través del Xbee) a formato decimal, de forma que, si se recibe el valor 0x32, se sepa que se trata del valor 50.

Por último, una vez se dispone del vector de 128 valores que indicaran los porcentajes de activación de las 128 salidas, se procede a la configuración de cada una de las salidas, calculando el valor de la señal PWM en puntos (entre 0 y 4095) de la siguiente forma:

$$valor\ PWM = floor\left(\frac{porcentaje \times 4095}{100}\right)$$

Siendo el valor obtenido, el valor que se enviará mediante la instrucción I2C correspondiente para su configuración.

4.5.3 Comprobación del funcionamiento.

Una vez cargado el código en el microprocesador, se montó un prototipo donde en lugar de usar los motores seleccionados, se colocaron leds, ya que, de esta forma, se podía observar las variaciones de intensidad en la luz de estos en función de la señal PWM configurada en cada salida.

Para comprobar el funcionamiento del código desarrollado, se montó un prototipo como el mostrado en la Figura 4-33, pero colocando un led en cada salida, teniendo en total 16 leds iluminados independientemente. De esta forma, el prototipo de prueba tiene la siguiente forma:

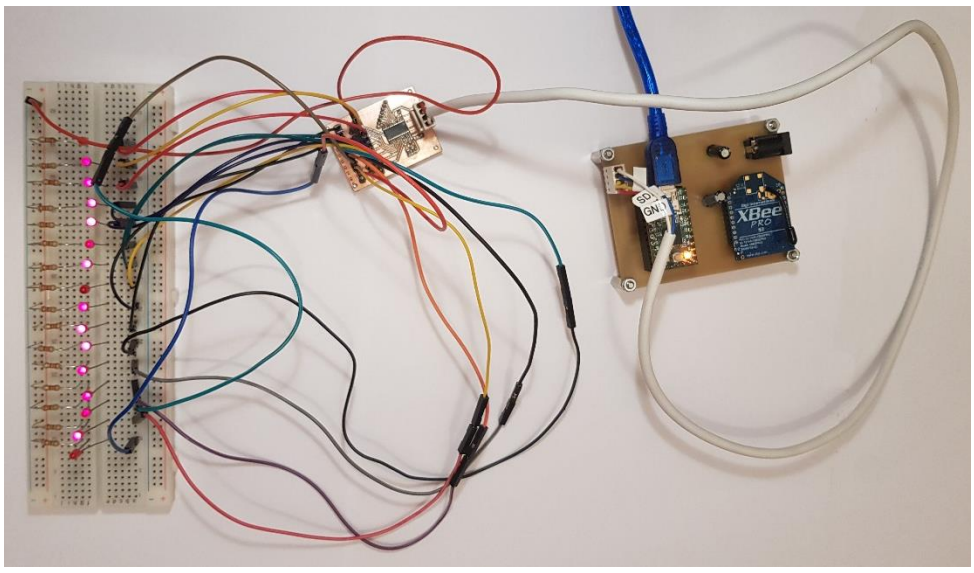


Figura 4-35: Prototipo de prueba final.

Nuevamente, una vez encendido dicho prototipo, a través de XCTU se envía un vector de datos de 16 valores, con los porcentajes de encendido de cada uno de los leds. A

continuación, puede verse un ejemplo de vector enviado con XCTU y el comportamiento de cada uno de los leds.

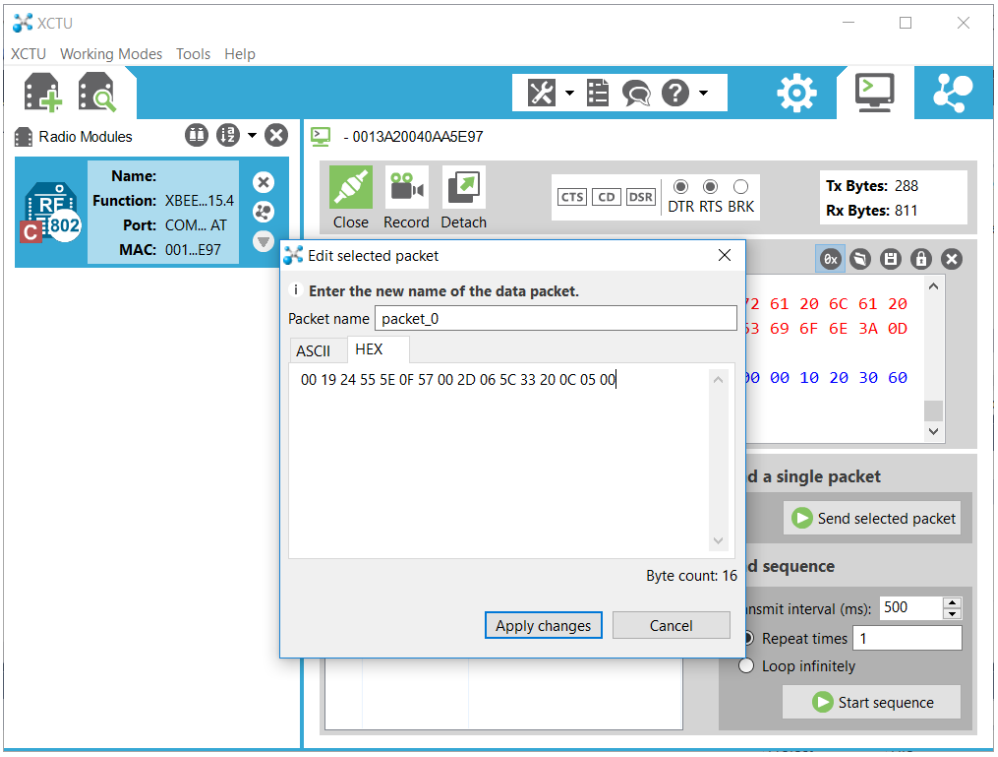


Figura 4-36: Ejemplo de configuración con XCTU

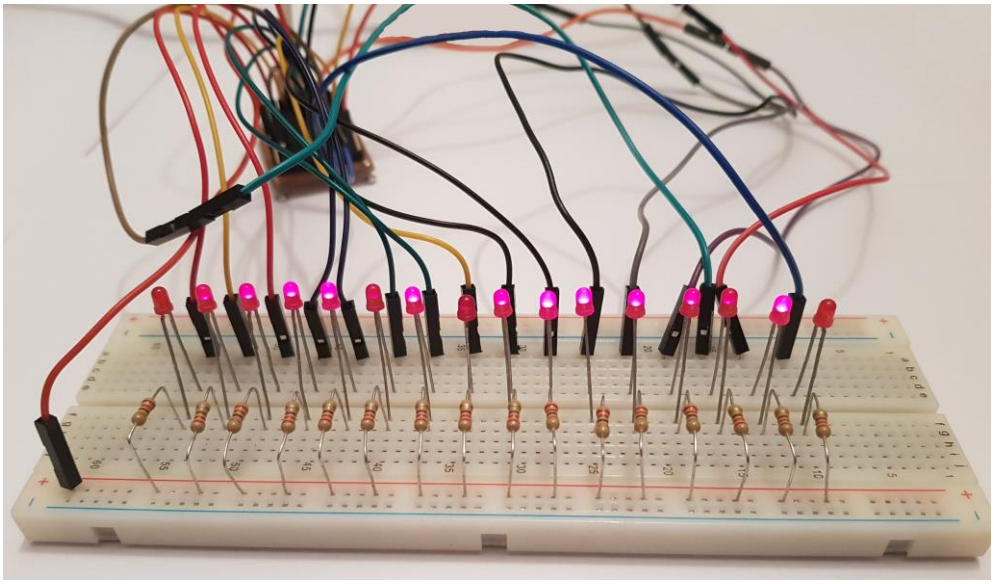


Figura 4-37: Ejemplo de configuración en cada salida.

Tal y como se puede observar en las imágenes anteriores (Figura 4-36 y Figura 4-37) el sistema funciona correctamente ya que en función de los datos enviados se obtiene una intensidad de encendido diferente en cada una de las salidas. De esta forma, una vez el sistema sea montado en el chaleco, podrá obtenerse diferentes intensidades de vibración en cada uno de los motores que formaran el chaleco, cumpliendo así con el objetivo principal de este TFM,

4.6 Diseño PCB Flexible.

Tal y como se puede observar en la Figura 4-35a pesar de haber reducido considerablemente el tamaño del prototipo, sigue habiendo gran cantidad de cables que recorrerán el chaleco para conectar cada uno de los motores. Por este motivo, se ha decidido desarrollar un nuevo diseño de un PCB que contendrá el driver con las diferentes conexiones de los motores.

El objetivo principal de este PCB es, al igual que el PCB anterior, conectar el driver PCA9685 a una señal I2C de entrada, pero además añadir diferentes puntos de conexión para cada uno de los motores formando así cada una de las filas que formará la matriz de motores del chaleco. Para ello, este PCB tendrá una forma alargada, de unos 30 cm, donde en lugar de tener cables para realizar las conexiones, se tendrán diferentes pistas a lo largo de la placa.

Por último, es muy importante que el PCB se fabrique sobre un material flexible, de forma que pueda ajustarse y amoldarse a la forma del pecho en la cual se colocará el chaleco con los diferentes motores.

De esta forma, el esquemático del circuito que formará este PCB, no es muy distinto al esquemático mostrado en .

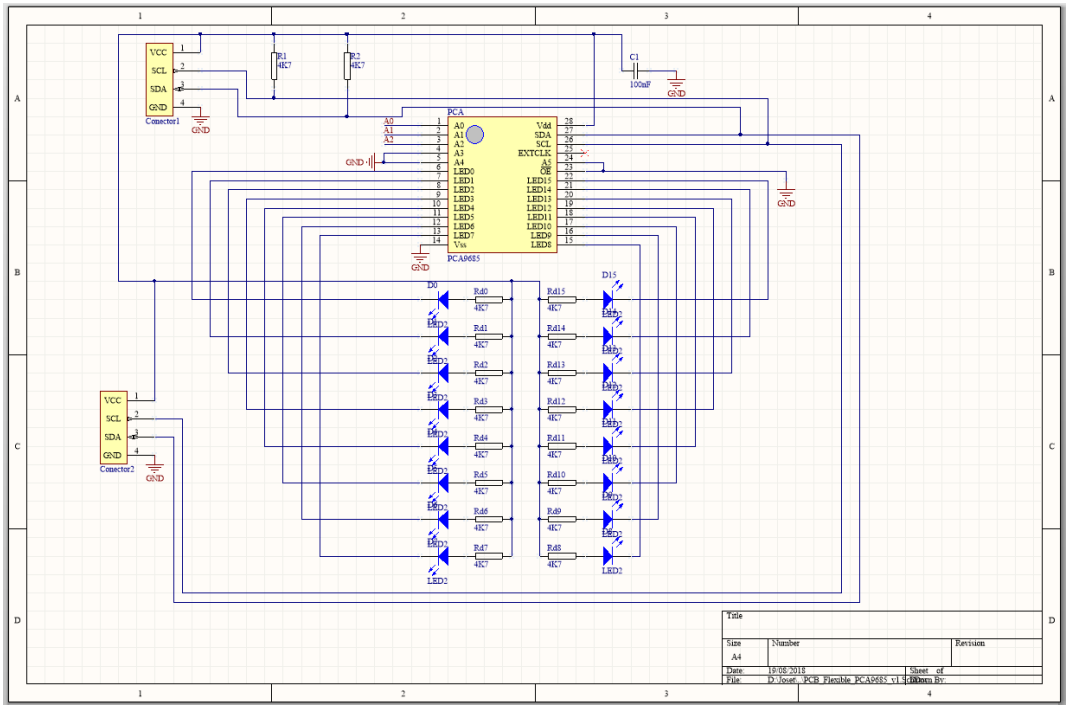


Figura 4-38: Esquemático para PCB Flexible.

Como se puede observar, en el esquemático anterior se han colocado leds con sus correspondientes resistencias en cada una de las salidas del driver. Esto se debe a que la idea principal de este PCB es que fuese realizado con un material flexible, pero debido a limitaciones de tiempo y fabricación de las cuales se dispone en la escuela, se optó por realizar este nuevo PCB sobre un material rígido como en las anteriores, y en lugar de colocar *Pads* de soldadura para cada motor, se utilizaran leds, con la finalidad de poder observar el funcionamiento de un prototipo y diseño final, donde más adelante este mismo diseño se pudiese realizar sobre un material flexible.

Por lo tanto, una vez explicado esto, el diseño del *Layout* de este nuevo PCB será el siguiente:

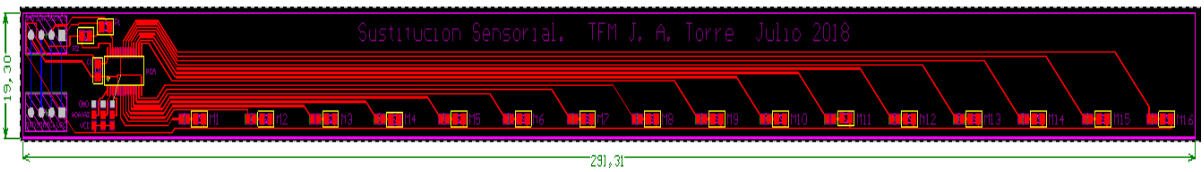


Figura 4-39: Layout PCB Flexible.

Tal y como se puede observar, cada uno de los motores, o leds para este prototipo, se soldarán a lo largo del PCB, eliminando así la necesidad de usar un par de cables para cada salida. Además, para poder seleccionar la dirección del esclavo sin necesidad de editar el diseño para cada uno, y que este fuese un diseño industrializable, se colocaron 3 pads de soldadura conectados a los tres puertos del chip en lo que se configura la dirección hardware del esclavo (los 3 bits menos significativos), de forma que mediante un punto de soldadura se pueda configurar cada entrada a "0" o a "1". Debido a que se definió inicialmente que, como mucho, sería necesario el uso de 8 esclavos diferentes, únicamente con configurar los 3 bits menos significativos del chip en lugar de los 6 bits disponibles, se tendrían direcciones hardware suficientes para cada uno de ellos.

A continuación, en la Figura 4-40 puede observarse el resultado final del PCB fabricado y como irán dispuestos los motores a lo largo de este, formando con cada uno una fila de la matriz que se colocará en el chaleco.

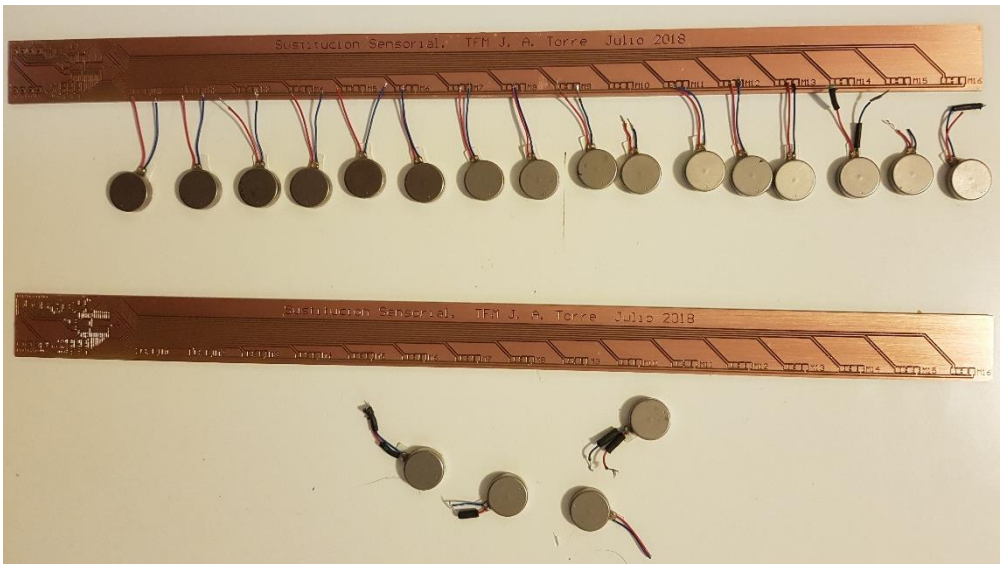


Figura 4-40: PCB "Flexible"

Finalmente, una vez desarrollados todos los elementos hardware que compondrán el prototipo y el software de control de este, se puede montar el prototipo final (Figura 4-41), el cual tendrá la siguiente forma:

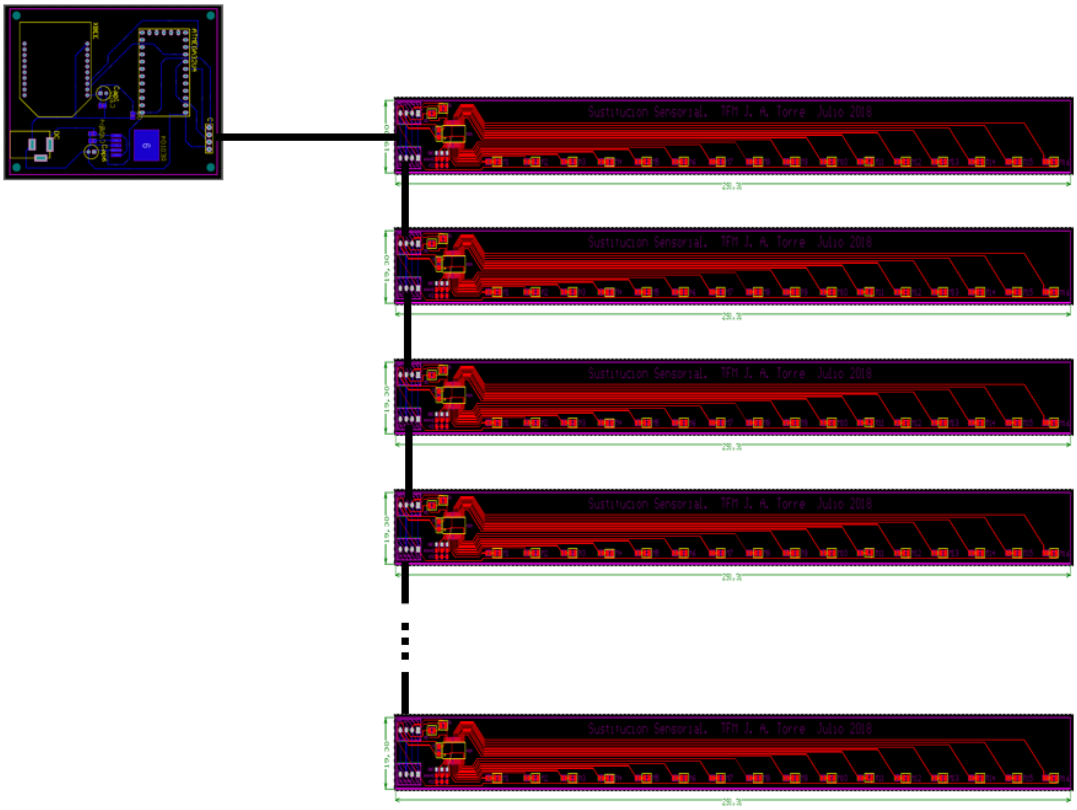


Figura 4-41: Prototipo Final.

Como se puede observar en la figura anterior (Figura 4-41) el prototipo estará formado por el módulo de comunicaciones y control (PCB con el microprocesador y Xbee), el cual se conectará a uno de los PCBs que contendrán el driver I2C mediante un cable con 4 puertos (SDA, SCL, Vcc y GND). A su vez, este primer PCB con el driver I2C, se conectará con un segundo driver, y así sucesivamente hasta conseguir el número de salidas deseadas (en este caso 128 salidas y por lo tanto 8 drivers). De esta forma, el diseño obtenido es un diseño modular, al cual se le pueden añadir tantos drivers, y por lo tanto salidas, como se deseen.

5 Conclusiones y trabajo futuro

5.1 Conclusiones

Como resultado de este TFM se ha obtenido un prototipo similar al desarrollado inicialmente por los investigadores del Laboratorio de Visión y Percepción de la Facultad de Psicología de la UAM, en el cual se han reducido considerablemente las dimensiones del mismo, además de eliminar la gran cantidad de cables que se utilizaban.

Por otra parte se ha simplificado considerablemente el prototipo, además de haber reducido el consumo de este. Esto se debe a que, en un inicio, como driver para los motores se usaban plataformas *Arduino,* capaces de activar como máximo 6 motores (22 Arduinos necesarios para 128 motores). En cambio, el driver PCA9685 es capaz de controlar 16 motores de manera independiente (8 drivers para 128 motores), y además no requiere de software adicional ya que son controlados a través del bus I2C.

Además de haber conseguido los objetivos principales de este TFM (reducir tamaño y cantidad de cables necesarios), también se han conseguido los objetivos secundarios que se plantearon. Como se ha podido observar se ha realizado un diseño modular y fácil de actualizar, de forma que, si por un casual ocurriese cualquier problema durante su uso en el microprocesador, se podría cambiar por otro igual sin necesidad de realizar ninguna tarea adicional, al igual que ocurre con el dispositivo Xbee. Además, al estar el prototipo separado en dos PCBs diferentes, uno de ellos contiene el dispositivo de comunicación inalámbrica y el microprocesador, y otro de ellos contiene el driver I2C, puede en cualquier momento intercambiarse por otros PCBs diferentes con otros dispositivos, siempre y cuando se realizase la comunicación I2C de manera adecuada.

En resumen, se ha diseñado, fabricado y programado un prototipo para el Laboratorio de Visión y Percepción de la Facultad de Psicología de la UAM, que cumple sin problemas los objetivos planteados, reduciendo considerablemente su tamaño y consumo, siendo un diseño modular, fácil de usar y de sustituir elementos dañados.

5.2 Trabajo futuro

Como trabajo a corto plazo se propone realizar el último de los PCBs descrito sobre un material flexible. Esta mejora dotaría al dispositivo de una mayor adaptabilidad a cualquier superficie, en este caso el pecho de una persona, y que por lo tanto pueda usarse en el chaleco necesario para los estudios del Laboratorio de Visión y Percepción de la Facultad de Psicología de la UAM.

Es importante destacar, que, para realizar este posible trabajo, ya se proporciona el diseño del PCB necesario para poder fabricarlo.

Como posible trabajo a largo plazo, se propone estudiar el funcionamiento y programación del dispositivo desarrollado por *Silicon Labs* basado en el protocolo Zigbee para comunicaciones inalámbricas con un microprocesador ARM Cortex M4 de 32 bits, el MGM12P, mediante el cual se reduciría considerablemente las dimensiones del PCB que contiene la plataforma Teensy 2.0 y el dispositivo Xbee, ya que con este único chip se podría realizar tanto la comunicación inalámbrica con el ordenador como el control de los diferentes

drivers I2C que controlaran los motores. Además, no solo mejoraría el tamaño del PCB, sino que las capacidades de procesamiento de este nuevo chip serían mucho mayores.

Por último, también se proponen realizar mejoras en el software que porta el prototipo, con la finalidad de mejorar el rendimiento, o añadir alguna funcionalidad extra al prototipo.

Referencias

[1] L. Lobo, D. Travieso, D. M. Jacobs, M. Rodger, C. M. Craig, “Sensory Substitution: Using a Vibrotactile Device to Orient and Walk to Targets”, Journal of Experimental Psychology: Applied, 2018, Vol. 24, No. 1, 108-124.

[2] P. Bach-y-Rita (1969), “Tactile sensory substitution studies”, Annals of the New York Academy of Sciences, 1013: 83-91.

[3] P. Bach-y-Rita, CC, Collins, F. Saunders, B. White, L. Scadden (1969). "Vision substitution by tactile the image projection". Nature. 221: 963–964.

[4] H. J. Bullinger, J. Ziegler, (1999) “Human-Computer Interaction: Communication, Cooperation, and Application Design”, Volumen 2, 1010-1015.

[5] C. Brill, J & Lawson, Ben & Rupert, Angus. (2014), “Tactile Situation Awareness System (TSAS) as a Compensatory Aid for Sensory Loss”, Proceedings of the Human Factors and Ergonomics Society Annual Meeting. 58. 1028-1032.

[6] Elliott, Linda & B F van Erp, Jan & S Redden, Elizabeth & Duistermaat, Maaike. (2010). “Field-Based Validation of a Tactile Navigation Device.” Haptics, IEEE Transactions on. 3. 78 – 87.

[7] L. A. Jones, N. B. Sarter, “Tactile Displays: Guidance for Their Design and Application”, HUMAN FACTORS, Vol. 50, No. 1, February 2008, pp. 90–111.

[8] S. Shoval, J. Borenstein, and Y. Koren, “Mobile robot obstacle avoidance in a computerized travel aid for the blind,” in Proc. 1994 IEEE Robot. Autom. Conf., San Diego, CA, May 8–13, pp. 2023–2029.

[9] P. B. L. Meijer. (1992, Feb.). An experimental system for auditory image representations. IEEE Trans. Biomed. Eng. [Online]. 39(2), pp. 112–121.

[10] G. Sainarayanan, R. Nagarajan, S. Yaacob, “Fuzzy image processing scheme for autonomous navigation of human blind,” Appl. Softw. Comput., vol. 7, no. 1, pp. 257–264, Jan. 2007.

[11] R. Audette, J. Balthazaar, C. Dunk, J. Zelek, “A stereo-vision system for the visually impaired,” Sch. Eng., Univ. Guelph, Guelph, ON, Canada, Tech. Rep. 2000-41x-1, 2000.

[12] S. Meers, K. Ward, “A substitute vision system for providing 3D perception and GPS navigation via electro-tactile stimulation,” presented at the 1st Int. Conf. Sens. Technol., Palmerston North, New Zealand, Nov. 21–23, 2005.

[13] L. A. Johnson, C. M. Higgins, “A navigation aid for the blind using tactile-visual sensory substitution,” in Proc. 28th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc., New York, 2006, pp. 6298–6292.

[14] S. Cardin, D. Thalmann, F. Vexo, “A wearable system for mobility improvement of visually impaired people,” Vis. Comput., vol. 23, no. 2, pp. 109–118, Jan. 2007.

[15] N. G. Bourbakis, D. Kavraki, “Intelligent assistants for handicapped people’s independence: Case study,” in Proc. 1996 IEEE Int. Joint Symp. Intell. Syst., Nov. 4–5, pp. 337–344.

[16] D. Dakopoulos, S. K. Boddhu, N. Bourbakis, “A 2D vibration array as an assistive device for visually impaired,” in Proc. 7th IEEE Int. Conf. Bioinf. Bioeng., Boston, MA, Oct. 14–17, 2007, vol. 1, pp. 930–937.

[17] D. Dakopoulos, N. Bourbakis, “Preserving visual information in low resolution images during navigation for visually impaired,” presented at the 1st Int. Conf. PErvasive Technol. Related Assist. Environ., Athens, Greece, Jul. 15–19, 2008.

[18] M. Adjouadi, “A man-machine vision interface for sensing the environment,” J. Rehabil. Res. Develop., vol. 29, no. 2, pp. 57–56, 1992.

[19] D. Yuan, R. Manduchi, “A tool for range sensing and environment discovery for the blind,” in Proc. 2004 Conf. Comput. Vis. Pattern Recogn. Workshop, Washington, DC, Jun. 27–Jul. 02, vol. 3, pp. 39–39.

[20] Bay Advanced Technologies Ltd. (2009, May 8). [Online]. Disponible en: http://www.batforblind.co.nz/index.htm

[21] BestPluton World Cie. (2009, May 8). [Online]. Disponible en: http://bestpluton.free.fr/EnglishMiniRadar.htm

[22] GDP Research, Australia. (2009, May 8). [Online]. Disponible en: http://www.gdp-research.com.au/

[23] Nurion-Raycal. [Online]. Disponible en: http://www.lasercane.com/

[24] Sound Foresight Ltd. (2009, May 8). [Online]. Disponible en: http://www.batcane.com/

[25] L. Cancar, A. Díaz, A. Barrientos, D. Travieso y D. M. Jacobs, “Tactile-Sight: A Sensory Substitution Device Base on Distance-Related Vibrotactile Flow”, International Journal of Advanced Robotic Systems, Regular Paper, Madrid, Febrero de 2013.

[26] Digi International, XBee/XBee-PRO S1 802.15.4 (Legacy). Disponible en: https://www.digi.com/resources/documentation/digidocs/pdfs/90000982.pdf

[27] Silicon Labs, MGM12P Mighty Gecko Multi-Protocol Wireless Mesh Module Data Sheet. Disponible en: https://www.silabs.com/documents/public/data-sheets/mgm12p-datasheet.pdf

[28] Atmel, ATmega32U4: 8-bit AVR Microcontroller with 32K Bytes of ISP Flash and USB Controller. Disponible en: http://www.atmel.com/Images/doc7766.pdf

[29] Beagle Board, BeagleBone Pocket: System Reference Manual. Disponible en: https://github.com/beagleboard/pocketbeagle/wiki/System-Reference-Manual#31_Whats_In_the_Package

[30] NXP, PCA9685 16-channel, 12-bit PWM Fm+ I2C-bus LED controller. Disponible en: https://cdn-shop.adafruit.com/datasheets/PCA9685.pdf

[31] Texas Instruments, DRV8830 Low-Voltage Motor Driver with Serial Interface. Disponible en: http://www.ti.com/lit/ds/symlink/drv8830.pdf

[32] Digi International, XCTU: Configuration and Test Utility Software. Disponible en: https://www.digi.com/resources/documentation/digidocs/pdfs/90001458-13.pdf

[33] Atmel, Atmel Studio 7 User Guide. Disponible en: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-Studio-7-User-Guide.pdf

[34] D. Paret, El bus I2C de la teoría a la práctica. Madrid: Paraninfo, D.L. 1995.

[35] Texas Instruments, Universal Synchronous Asynchronous Receive/Transmit USART. Disponible en: https://www.ti.com/sc/docs/products/micro/msp430/userguid/ag_12.pdf

Anexos

A Registros de control del bus TWI.

A.1 Registro TWI Bit Rate (TWBR).

Este registro (del bit 7 al 0) (Figura 0-1) selecciona el factor de división para el generador de tasa de bits. El generador de tasa de bits es un divisor de frecuencia que genera la frecuencia de reloj encargada de la sincronización (SCL), la cual se calcula de la siguiente forma:

$$Frecuencia\ SCL = \frac{Frecuencia\ de\ Reloj\ de\ la\ CPU}{16 + 2 * (TWBR) * 4^{TWPS}}$$

Donde la velocidad del reloj de la CPU son 16MHz, TWBR es el valor configurado en este registro, y TWPS es un valor de escala, el cual se configura en el registro TWSR que se verá más adelante.

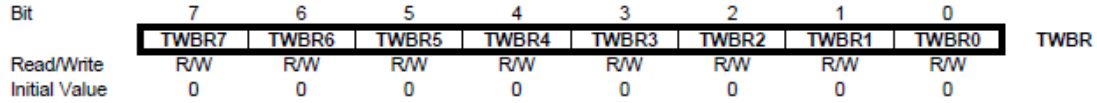


Figura 0-1: Registro TWBR.

A.2 Registro TWI Control (TWCR).

Este registro se usa para controlar el funcionamiento del TWI. Se usa para habilitar el bus, enviar una condición de START, generar un reconocimiento del receptor, generar una condición de STOP y controlar la detención del bus mientras los datos se escriben en el registro TWDR. En este registro, cada bit tiene una función diferente (Figura 0-2)

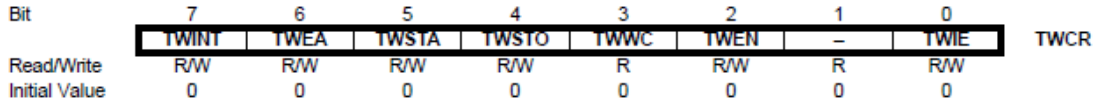


Figura 0-2: Registro TWCR.

- **TWINT - TWI Interrupt Flag (Bit 7):** Este bit lo establece el hardware cuando el TWI ha terminado su trabajo actual y espera la respuesta del software de la aplicación.
- **TWEA – TWI Enable Acknowledge Bit (Bit 6):** Este bit controla la generación del pulso ACK. Si el bit se pone a "1", el pulso ACK se genera en el bus TWI, siempre y cuando se haya recibido la dirección del esclavo, se haya recibido una llamada general o se hayan recibido los datos correctamente.
- **TWSTA – TWI START Condition Bit (Bit 5):** Este bit se pone a "1" cuando el dispositivo va a funcionar como maestro. Si el bus está disponible se genera una condición de START, en caso contrario, espera a que haya una condición de STOP y seguidamente manda un START. Una vez se manda la condición START, este bit debe ponerse a "0".

- **TWSTO – TWI STOP Condition Bit (Bit 4):** Si el dispositivo está configurado como Maestro, al establecer este bit a “1”, se envía una condición de STOP, e inmediatamente se pone a “0”. Si, por el contrario, está configurado como Esclavo, este bit se puede usar para recuperarse de una condición de error.
- **TWWC – TWI Write Collision Flag (Bit 3):** Este bit se pone a “1” cuando se intenta escribir en TWDR y el registro TWINT está bajo, y se pone a “0” si TWINT esta alto.
- **TWEN – TWI Enable Bit (Bit 2):** Este bit habilita la operación TWI. Si está a “1”, el TWI toma el control de los pines SCL y SDA. Si por el contrario está a “0”, el TWI se apaga y todas las transmisiones finalizan, aunque se encuentre una operación en curso.
- **Res – Bit Reservado (Bit 1):** Este bit está reservado y siempre se leerá como un “0”.
- **TWIE: TWI Interrupt Enable (Bit 0):** Si este bit está a “1”, se activa la solicitud de interrupción TWI mientras el indicador TWINT este a “1”.

A.3 Registro TWI Status (TWSR):

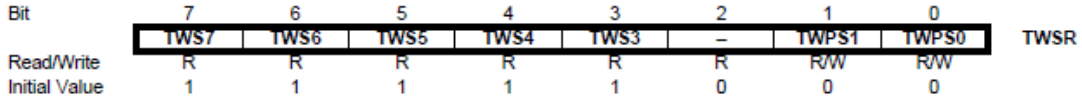


Figura 0-3: Registro TWSR.

- **TWS – TWI Status (Bits del 7 al 3):** Estos 5 bits reflejan el estado del bus TWI.
- **Res – Bit Reservado (Bit 2).**
- **TWPS – TWI Prescaler Bits (Bits 1 y 0):** Estos bits se pueden leer y escribir, y controlan la tasa de escala de la frecuencia SCL (Tabla 3).

Tabla 3. Valores de escala del bit TWPS.

TWPS1	TWPS0	Valor de Escala
0	0	1
0	1	4
1	0	16
1	1	64

A.4 Registro TWI Data (TWDR):

Este registro (Figura 0-4) contiene los datos en el siguiente Byte si se está transmitiendo, y el valor del ultimo Byte si se está recibiendo. Los datos permanecen estables siempre y cuando TWINT sea “1”. Mientras los datos se están transmitiendo, los datos se cambian simultáneamente.

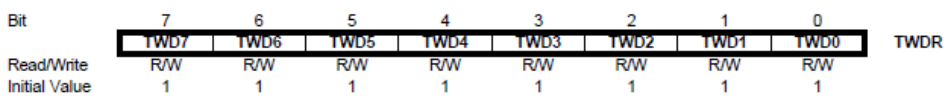


Figura 0-4: Registro TWDR.

A.5 Registro TWI Address (Esclavo) (TWAR):

Este registro (Figura 0-5) debe cargarse con la dirección del esclavo (en los 7 bits más significativos) a la que responderá el TWI cuando se programe como transmisor o receptor esclavo.

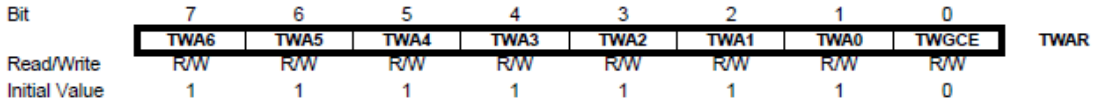


Figura 0-5: Registro TWAR.

- **TWA – TWI Address (Esclavo) (Bits del 7 al 1):** Estos 7 bits constituyen la dirección del esclavo del TWI.
- **TWGCE – TWI General Call Recognition Enable Bit (Bit 0):** Si se pone a "1", este bit permite el reconocimiento de una llamada general a traves del bus.

A.6 Registro TWI Address Mask Register (TWAMR):

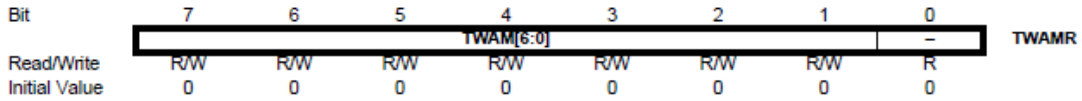


Figura 0-6: Registro TWAMR.

- **TWAM – TWI Adrress Mask (Bits del 7 al 1):** Estos bits se pueden cargar con una máscara de Dirección de esclavo de 7 bits. Cada uno de los bits en TWAMR puede enmascarar (deshabilitar) el bit de dirección correspondiente en el Registro de direcciones TWI (TWAR). La siguiente figura (Figura 0-7) muestra el funcionamiento de estos bits:

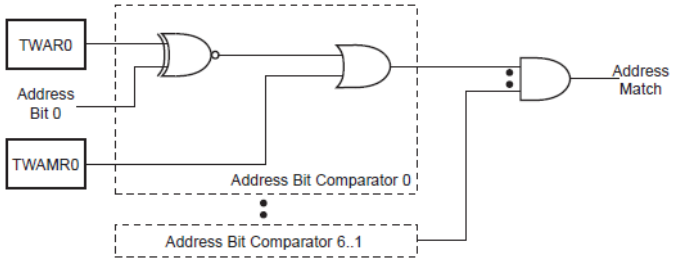


Figura 0-7: Circuito lógico de la máscara TWAM.

- **Res – Bit Reservado (Bit 0).**

B Registros de control del módulo USART.

B.1 USART I/O Data Register (UDRn).

El buffer de recepción de datos (RBX) y el buffer de transmisión de datos (TXB) comparten la misma dirección en el registro UDRn. Cuando los datos se escriben en el búfer de transmisión y el transmisor está habilitado (bandera UDREn en el Registro UCSRnA), el transmisor cargará los datos en el *Transmit Shift Register*, siempre y cuando el *Shift Register* esté vacío. Luego, los datos se transmitirán en serie en el pin TxDn.

En cambio, el buffer de recepción consiste en una FIFO (*First In First Out*) de dos niveles. La FIFO cambiará su estado siempre que se acceda al buffer de recepción, por lo que no utilizará instrucciones de Lectura-Modificación-Escritura.

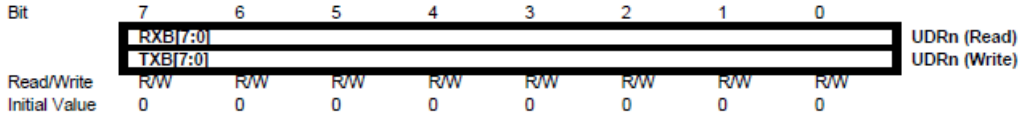


Figura 0-8: Registro UDRn.

B.2 USART Control and Status Register A (UCSRnA).

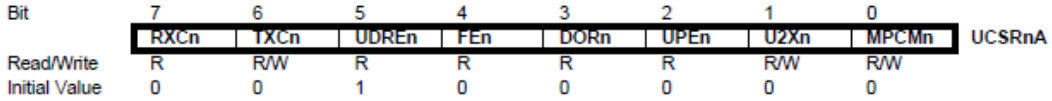


Figura 0-9: Registro UCSRnA.

- **RXCn – USART Receive Complete (Bit 7):** Este bit se pone a "1" cuando hay datos no leídos en el buffer de recepción y se pone a "0" cuando el buffer de recepción está vacío. Si el receptor esta deshabilitado, el buffer de recepción se vaciará, y, en consecuencia, este bit se pondrá a "0".
- **TXCn – USART Transmit Complete (Bit 6):** Este bit se pone a "1" cuando el contenido del *Transmit Shift Register* se ha enviado y no hay datos nuevos actualmente presentes en el buffer de transmisión. Este bit se borra automáticamente cuando se ejecuta una interrupción de transmisión completa.
- **UDREn – USART Data Register Empty (Bit 5):** Este bit indica si el buffer de transmisión (UDRn) está listo para recibir nuevos datos. Si UDREn es uno, el buffer está vacío, y, por lo tanto, listo para ser escrito.
- **FEn – Frame Error (Bit 4):** Este bit se establece a "1" si el siguiente carácter en el buffer de recepción tuvo un *Frame Error* cuando se recibió, es decir, cuando el primer bit de parada del siguiente carácter en el búfer de recepción es "0".
- **DORn – Data OverRun (Bit 3):** Este bit se establece a "1" si se detecta una condición de *Data OverRun*. Un *Data Overrun* ocurre cuando el buffer de recepción está lleno (dos caracteres), un nuevo carácter está esperando en el *Receive Shift Register*, y se detecta un nuevo bit de inicio.

- **UPEn – USART Parity Error (Bit 2):** Este bit se establece a “1” si el siguiente carácter en el búfer de recepción tuvo un error de paridad cuando se recibió, y, además, la verificación de paridad se habilitó (UPMn1 = 1).
- **U2Xn – Double the USART Transmissio Speed (Bit 1):** Este bit solo tiene efecto cuando se trabaja en modo asíncrono (Cuando se trabaja en modo síncrono, este bit debe escribirse a “0”). Escribir este bit a uno reducirá la velocidad en baudios de 16 a 8, doblando efectivamente la tasa de transferencia para la comunicación asíncrona.
- **MPCMn – Multi-processor Communication Mode (Bit 0):** Este bit habilita el modo de comunicación multiprocesador. Cuando el bit MPCMn se pone a “1”, se ignorarán todas las tramas entrantes recibidas por el Receptor USART que no contengan información de dirección. El transmisor no se ve afectado por la configuración de este bit.

B.3 USART Control and Status Register B (UCSRnB).

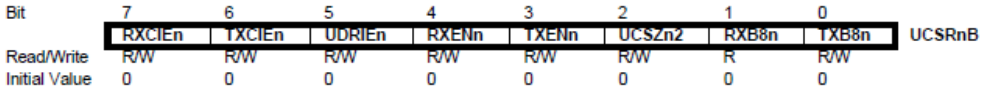


Figura 0-10: Registro UCSRnB.

- **RXCIEn – RX Complete Interrupt Enable n (Bit 7):** Escribir este bit a “1” habilita la interrupción en la bandera RXCn.
- **TXCIEn – TX Complete Interrupt Enable n (Bit 6):** Escribir este bit a “1” habilita la interrupción en la bandera TXCn.
- **UDRIEn – USART Data Register Empty Interrupt Enable n (Bit 5):** Escribir este bit a “1” habilita la interrupción en la bandera UDREn.
- **RXENn – Receiver Enable n (Bit 4):** Escribir este bit a “1” habilita el receptor. Este anulará el uso del pin RxDn cuando esté habilitado.
- **TXENn – Transmitter Enable n (Bit 3):** Escribir este bit a “1” habilita el transmisor. El transmisor anulará el uso del pin TxDn cuando esté habilitado. La desactivación del transmisor (escribir TXENn a “0”) no entrará en vigencia hasta que se completen las transmisiones en curso y pendientes, es decir, cuando se transmita y cambie el registro de transmisión.
- **UCSZn2 – Character Size n (Bit 2):** El bit UCSZn2 combinados con los bits UCSZn1 y UCSZn0 en UCSRnC establecen el número de bits de en un frame que utilizan el receptor y el transmisor.
- **RXB8n – Receive Data Bit 8 n (Bit 1):** Este se corresponde con el noveno bit de datos del carácter recibido cuando se opera con frames serie de 9 bits. Debe leerse antes de leer los 8 bits restantes.
- **TXB8n – Transmit Data Bit 8 n (Bit 0):** Este se corresponde con el noveno bit de datos del carácter que se transmitirá cuando se opera con frames serie de 9 bits. Debe escribirse antes de escribir los 8 bits restantes.

B.4 USART Control and Status Register n C (UCSRnC).

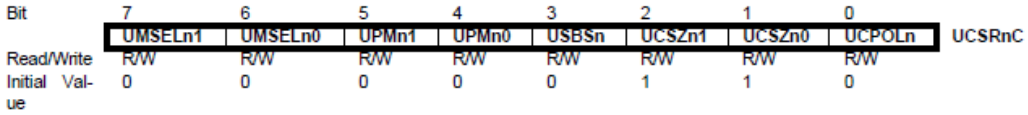


Figura 0-11: Registro UCSRnC.

- **UMSELn1:0 – USART Mode Select (Bits 7 y 6):** Estos bits seleccionan el modo de operación de USARTn.

Tabla 4. Configuración del bit UMSELn1:0

UMSELn1	UMSELn0	Mode
0	0	USART Asíncrono
0	1	USART Síncrono
1	0	(Reservado)
1	1	Maestro SPI.

- **UPMn1:0 – Parity Mode (Bits 5 y 4):** Estos bits habilitan y establecen el tipo de generación y verificación de paridad.

Tabla 5. Configuración del bit UPMn1:0.

UPMn1	UPMn0	Parity Mode
0	0	Deshabilitado
0	1	(Reservado)
1	0	Habilitado, Paridad Par
1	1	Habilitado, Paridad Impar

- **USBSn – Stop Bit Select (Bit 3):** Este bit selecciona la cantidad de bits de parada que insertará el transmisor. El receptor ignora esta configuración. Si este bit vale “0”, usará 1 bit de parada, y si este bit vale “1”, usará 2 bits.
- **UCSZn1:0 – Character Size (Bits 2 y 1):** Los bits UCSZn1: 0 combinados con el bit UCSZn2 en UCSRnB, establecen el número de bits de datos en un Frame que usan el receptor y el transmisor.

Tabla 6. Configuración de los bits UCSZn.

UCSZn2	UCSZn1	UCSZn0	Tamaño de Caracter
0	0	0	5 bits
0	0	1	6 bits
0	1	0	7 bits
0	1	1	8 bits
1	0	0	Reservado
1	0	1	Reservado
1	1	0	Reservado
1	1	1	9 bits

- **UCPOLn – Clock Polarity (Bit 0):** Este bit se usa solo para el modo síncrono. Escribir este bit a cero cuando se use el modo asíncrono. El bit UCPOLn establece la relación entre el cambio de salida de datos y la muestra de entrada de datos, y el reloj (XCKn).

B.5 USART Control and Status Register n D (UCSRnD).

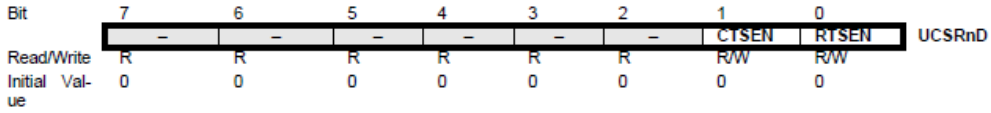


Figura 0-12: Registro UCSRnD.

- **Bits Reservados (Bits del 7 al 2).**
- **CTSEN – UART CTS Signal Enable (Bit 1):** La transmisión se habilitará solo si la entrada CTS es 0.
- **RTSEN – UART RTS Signal Enable (Bit 0):** En este caso, la línea RTS aumentará automáticamente cuando la FIFO esté lleno.

B.6 USART Baud Rate Registers – UBRRLn y UBRRHn.

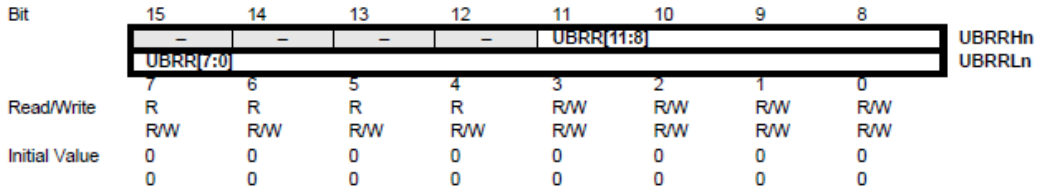


Figura 0-13 Registros UBRRLn y UBRRHn.

- **Bits Reservados (Bits del 15 al 12).**
- **UBRR11:0 – USART Baud Rate Register (Bits del 11 al 0):** Este es un registro de 12 bits que contiene la tasa de baudios USART. El UBRRH contiene los cuatro bits más significativos, y el UBRRL contiene los ocho bits menos significativos de la tasa de baudios. Las transmisiones en curso por el transmisor y el receptor se dañarán si se cambia la velocidad en baudios.

C Tabla de Registros de Control del PCA9685.

Register# (decimal)	Register# (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Name	Type	Function
0	00	0	0	0	0	0	0	0	0	MODE1	read/write	Mode register 1
1	01	0	0	0	0	0	0	0	1	MODE2	read/write	Mode register 2
2	02	0	0	0	0	0	0	1	0	SUBADR1	read/write	I²C-bus subaddress 1
3	03	0	0	0	0	0	0	1	1	SUBADR2	read/write	I²C-bus subaddress 2
4	04	0	0	0	0	0	1	0	0	SUBADR3	read/write	I²C-bus subaddress 3
5	05	0	0	0	0	0	1	0	1	ALLCALLADR	read/write	LED All Call I²C-bus address
6	06	0	0	0	0	0	1	1	0	LED0_ON_L	read/write	LED0 output and brightness control byte 0
7	07	0	0	0	0	0	1	1	1	LED0_ON_H	read/write	LED0 output and brightness control byte 1
8	08	0	0	0	0	1	0	0	0	LED0_OFF_L	read/write	LED0 output and brightness control byte 2
9	09	0	0	0	0	1	0	0	1	LED0_OFF_H	read/write	LED0 output and brightness control byte 3
10	0A	0	0	0	0	1	0	1	0	LED1_ON_L	read/write	LED1 output and brightness control byte 0
11	0B	0	0	0	0	1	0	1	1	LED1_ON_H	read/write	LED1 output and brightness control byte 1
12	0C	0	0	0	0	1	1	0	0	LED1_OFF_L	read/write	LED1 output and brightness control byte 2
13	0D	0	0	0	0	1	1	0	1	LED1_OFF_H	read/write	LED1 output and brightness control byte 3
14	0E	0	0	0	0	1	1	1	0	LED2_ON_L	read/write	LED2 output and brightness control byte 0
15	0F	0	0	0	0	1	1	1	1	LED2_ON_H	read/write	LED2 output and brightness control byte 1
16	10	0	0	0	1	0	0	0	0	LED2_OFF_L	read/write	LED2 output and brightness control byte 2
17	11	0	0	0	1	0	0	0	1	LED2_OFF_H	read/write	LED2 output and brightness control byte 3
18	12	0	0	0	1	0	0	1	0	LED3_ON_L	read/write	LED3 output and brightness control byte 0
19	13	0	0	0	1	0	0	1	1	LED3_ON_H	read/write	LED3 output and brightness control byte 1
20	14	0	0	0	1	0	1	0	0	LED3_OFF_L	read/write	LED3 output and brightness control byte 2
21	15	0	0	0	1	0	1	0	1	LED3_OFF_H	read/write	LED3 output and brightness control byte 3

Register# (decimal)	Register# (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Name	Type	Function
22	16	0	0	0	1	0	1	1	0	LED4_ON_L	read/write	LED4 output and brightness control byte 0
23	17	0	0	0	1	0	1	1	1	LED4_ON_H	read/write	LED4 output and brightness control byte 1
24	18	0	0	0	1	1	0	0	0	LED4_OFF_L	read/write	LED4 output and brightness control byte 2
25	19	0	0	0	1	1	0	0	1	LED4_OFF_H	read/write	LED4 output and brightness control byte 3
26	1A	0	0	0	1	1	0	1	0	LED5_ON_L	read/write	LED5 output and brightness control byte 0
27	1B	0	0	0	1	1	0	1	1	LED5_ON_H	read/write	LED5 output and brightness control byte 1
28	1C	0	0	0	1	1	1	0	0	LED5_OFF_L	read/write	LED5 output and brightness control byte 2
29	1D	0	0	0	1	1	1	0	1	LED5_OFF_H	read/write	LED5 output and brightness control byte 3
30	1E	0	0	0	1	1	1	1	0	LED6_ON_L	read/write	LED6 output and brightness control byte 0
31	1F	0	0	0	1	1	1	1	1	LED6_ON_H	read/write	LED6 output and brightness control byte 1
32	20	0	0	1	0	0	0	0	0	LED6_OFF_L	read/write	LED6 output and brightness control byte 2
33	21	0	0	1	0	0	0	0	1	LED6_OFF_H	read/write	LED6 output and brightness control byte 3
34	22	0	0	1	0	0	0	1	0	LED7_ON_L	read/write	LED7 output and brightness control byte 0
35	23	0	0	1	0	0	0	1	1	LED7_ON_H	read/write	LED7 output and brightness control byte 1
36	24	0	0	1	0	0	1	0	0	LED7_OFF_L	read/write	LED7 output and brightness control byte 2
37	25	0	0	1	0	0	1	0	1	LED7_OFF_H	read/write	LED7 output and brightness control byte 3
38	26	0	0	1	0	0	1	1	0	LED8_ON_L	read/write	LED8 output and brightness control byte 0
39	27	0	0	1	0	0	1	1	1	LED8_ON_H	read/write	LED8 output and brightness control byte 1
40	28	0	0	1	0	1	0	0	0	LED8_OFF_L	read/write	LED8 output and brightness control byte 2
41	29	0	0	1	0	1	0	0	1	LED8_OFF_H	read/write	LED8 output and brightness control byte 3

Register# (decimal)	Register# (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Name	Type	Function
42	2A	0	0	1	0	1	0	1	0	LED9_ON_L	read/write	LED9 output and brightness control byte 0
43	2B	0	0	1	0	1	0	1	1	LED9_ON_H	read/write	LED9 output and brightness control byte 1
44	2C	0	0	1	0	1	1	0	0	LED9_OFF_L	read/write	LED9 output and brightness control byte 2
45	2D	0	0	1	0	1	1	0	1	LED9_OFF_H	read/write	LED9 output and brightness control byte 3
46	2E	0	0	1	0	1	1	1	0	LED10_ON_L	read/write	LED10 output and brightness control byte 0
47	2F	0	0	1	0	1	1	1	1	LED10_ON_H	read/write	LED10 output and brightness control byte 1
48	30	0	0	1	1	0	0	0	0	LED10_OFF_L	read/write	LED10 output and brightness control byte 2
49	31	0	0	1	1	0	0	0	1	LED10_OFF_H	read/write	LED10 output and brightness control byte 3
50	32	0	0	1	1	0	0	1	0	LED11_ON_L	read/write	LED11 output and brightness control byte 0
51	33	0	0	1	1	0	0	1	1	LED11_ON_H	read/write	LED11 output and brightness control byte 1
52	34	0	0	1	1	0	1	0	0	LED11_OFF_L	read/write	LED11 output and brightness control byte 2
53	35	0	0	1	1	0	1	0	1	LED11_OFF_H	read/write	LED11 output and brightness control byte 3
54	36	0	0	1	1	0	1	1	0	LED12_ON_L	read/write	LED12 output and brightness control byte 0
55	37	0	0	1	1	0	1	1	1	LED12_ON_H	read/write	LED12 output and brightness control byte 1
56	38	0	0	1	1	1	0	0	0	LED12_OFF_L	read/write	LED12 output and brightness control byte 2
57	39	0	0	1	1	1	0	0	1	LED12_OFF_H	read/write	LED12 output and brightness control byte 3
58	3A	0	0	1	1	1	0	1	0	LED13_ON_L	read/write	LED13 output and brightness control byte 0
59	3B	0	0	1	1	1	0	1	1	LED13_ON_H	read/write	LED13 output and brightness control byte 1
60	3C	0	0	1	1	1	1	0	0	LED13_OFF_L	read/write	LED13 output and brightness control byte 2
61	3D	0	0	1	1	1	1	0	1	LED13_OFF_H	read/write	LED13 output and brightness control byte 3

Register# (decimal)	Register# (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Name	Type	Function
62	3E	0	0	1	1	1	1	1	0	LED14_ON_L	read/write	LED14 output and brightness control byte 0
63	3F	0	0	1	1	1	1	1	1	LED14_ON_H	read/write	LED14 output and brightness control byte 1
64	40	0	1	0	0	0	0	0	0	LED14_OFF_L	read/write	LED14 output and brightness control byte 2
65	41	0	1	0	0	0	0	0	1	LED14_OFF_H	read/write	LED14 output and brightness control byte 3
66	42	0	1	0	0	0	0	1	0	LED15_ON_L	read/write	LED15 output and brightness control byte 0
67	43	0	1	0	0	0	0	1	1	LED15_ON_H	read/write	LED15 output and brightness control byte 1
68	44	0	1	0	0	0	1	0	0	LED15_OFF_L	read/write	LED15 output and brightness control byte 2
69	45	0	1	0	0	0	1	0	1	LED15_OFF_H	read/write	LED15 output and brightness control byte 3
...	reserved for future use											
250	FA	1	1	1	1	1	0	1	0	ALL_LED_ON_L	write/read zero	load all the LEDn_ON registers, byte 0
251	FB	1	1	1	1	1	0	1	1	ALL_LED_ON_H	write/read zero	load all the LEDn_ON registers, byte 1
252	FC	1	1	1	1	1	1	0	0	ALL_LED_OFF_L	write/read zero	load all the LEDn_OFF registers, byte 0
253	FD	1	1	1	1	1	1	0	1	ALL_LED_OFF_H	write/read zero	load all the LEDn_OFF registers, byte 1
254	FE	1	1	1	1	1	1	1	0	PRE_SCALE[1]	read/write	prescaler for PWM output frequency
255	FF	1	1	1	1	1	1	1	1	TestMode[2]	read/write	defines the test mode to be entered
...	All further addresses are reserved for future use; reserved addresses will not be acknowledged.											

[1] Writes to PRE_SCALE register are blocked when SLEEP bit is logic 0 (MODE 1).

[2] Reserved. Writes to this register may cause unpredictable results.

D Códigos Fuente.

En este anexo se presentan los códigos fuente de todas las funciones que componen el software creado para el funcionamiento del prototipo.

En la cabecera de cada función se puede encontrar el nombre del archivo, una pequeña descripción, la fecha de la última modificación y autor del archivo.

```
/************************************************************
** ARCHIVO: main.c                                         **
**                                                         **
** Archivo que contiene el código principal del software. **
**                                                         **
** AUTOR: Jose Antonio Torre Albarsanz                     **
** FECHA: 20/05/2018                                       **
**                                                         **
************************************************************/

#include <avr/io.h>
#include "i2c.h"
#include "Xbee.h"
#include <util/delay.h>
#include "PCA9685.h"
#include <stdio.h>
#include <string.h>

int hex2ascii(uint8_t hex)
{
        int valor = 0;

        if(hex == 0x30){
                valor = 0;
                //uart_putchar(0x30);
        }
        else if(hex == 0x31){
                valor = 1;
                //uart_putchar(0x31);
        }
        else if(hex == 0x32){
                valor = 2;
                //uart_putchar(0x32);
        }
        else if(hex == 0x33){
                valor = 3;
                //uart_putchar(0x33);
        }
        else if(hex == 0x34){
                valor = 4;
                //uart_putchar(0x34);
        }
        else if(hex == 0x35){
                valor = 5;
                //uart_putchar(0x35);
        }
        else if(hex == 0x36){
                valor = 6;
                //uart_putchar(0x36);
        }
        else if(hex == 0x37){
                valor = 7;
                //uart_putchar(0x37);
        }
        else if(hex == 0x38){
                valor = 8;
                //uart_putchar(0x38);
        }
        else if(hex == 0x39){
                valor = 9;
```

```
                //uart_putchar(0x39);
        }
        else{
                valor = 0;
        }

        return valor;
}

void encendido(int dimension, uint8_t addrMaster1){
        for(int j = 0; j < dimension/2; j++){
                setPWM(addrMaster1, j, 4096, 0);
        }

        for(int times = 0; times < 3; times++){
                for (int j = 0; j < dimension/2; j++){
                        if (j==0){
                                setPWM(addrMaster1, j, 0, 0);
                                setPWM(addrMaster1, j+1, 4096, 0);
                        }
                        else{
                                setPWM(addrMaster1, j-1, 4096, 0);
                                setPWM(addrMaster1, j, 0, 0);
                                setPWM(addrMaster1, j+1, 4096, 0);
                        }
                        _delay_ms(50);
                }
                for (int i = 0; i < dimension/2; i++){
                        if (i==15){
                                setPWM(addrMaster1, 15-i, 0, 0);
                                setPWM(addrMaster1, 15-i+1, 4096, 0);
                        }
                        else{
                                setPWM(addrMaster1, 15-i-1, 4096, 0);
                                setPWM(addrMaster1, 15-i, 0, 0);
                                setPWM(addrMaster1, 15-i+1, 4096, 0);
                        }
                        _delay_ms(50);
                }
        }

        for(int j = 0; j < dimension/2; j++){
                setPWM(addrMaster1, j, 4096, 0);
        }
}

int main(void)
{
        //ENCENDIDO DEL LED DEL MICRO.
        LED_CONFIG;
        LED_ON;

        //INICIALIZACIÓN DE DIRECCIONES DE LOS ESCLAVOS.
        uint8_t addrMaster1 = 0x4D;
        uint8_t addrMaster2 = 0x4D;
        uint8_t addrMaster3 = 0x4D;
        uint8_t addrMaster4 = 0x4D;
        uint8_t addrMaster5 = 0x4D;
        uint8_t addrMaster6 = 0x4D;
        uint8_t addrMaster7 = 0x4D;
        uint8_t addrMaster8 = 0x4D;

        //INICIALIZACIÓN DEL DRIVER I2C: (CAMBIAR CUANDO AÑADAMOS MAS ESCLAVOS)
        begin(addrMaster1);
        setPWMFreq(addrMaster1, 1600);
        begin(addrMaster2);
        setPWMFreq(addrMaster2, 1600);
        begin(addrMaster3);
        setPWMFreq(addrMaster3, 1600);
        begin(addrMaster4);
        setPWMFreq(addrMaster4, 1600);
        begin(addrMaster5);
        setPWMFreq(addrMaster5, 1600);
        begin(addrMaster6);
```

```
setPWMFreq(addrMaster6, 1600);
begin(addrMaster7);
setPWMFreq(addrMaster7, 1600);
begin(addrMaster8);
setPWMFreq(addrMaster8, 1600);

//INICIALIZACION VARIABLES.
int dimension = 32;
uint8_t array1[dimension];
int valores[dimension];
int valor = 0;
int pwm;

//INICIALIZACIÓN DEL XBEE:
CPU_PRESCALE(0);  // run at 16 MHz
uart_init(BAUD_RATE);

//SECUENCIA DE ENCENDIDO.
uart_print("\r\nIniciando Dispositivo...\r\n");
encendido(dimension, addrMaster1);
encendido(dimension, addrMaster2);
encendido(dimension, addrMaster3);
encendido(dimension, addrMaster4);
encendido(dimension, addrMaster5);
encendido(dimension, addrMaster6);
encendido(dimension, addrMaster7);
encendido(dimension, addrMaster8);
uart_print("\r\nListo para la comunicacion:\r\n");

while(1){

        if (uart_available() >= dimension) {

                for(int i = 0; i < dimension; i++){
                        array1[i] = uart_getchar();
                        valores[i] = hex2ascii(array1[i]);
                }

                /*for(int pin=0; pin < dimension/2; pin++){
                        int a = valores[pin];
                        int b = valores[pin + dimension/2];*/
                int pin = 0;
                for(int i=0; i < dimension; i = i + 2){
                        int a = valores[i];
                        int b = valores[i + 1];
                        uint8_t addrM = 0x4D;

                        valor = 100 - 10*a + b;
                        pwm = floor(valor*40.95);

                        if(pin <= 15){
                                addrM = addrMaster1;
                        }
                        else if(pin >= 16 && pin <= 31){
                                addrM = addrMaster2;
                        }
                        else if(pin >= 32 && pin <= 47){
                                addrM = addrMaster3;
                        }
                        else if(pin >= 48 && pin <= 63){
                                addrM = addrMaster4;
                        }
                        else if(pin >= 64 && pin <= 79){
                                addrM = addrMaster5;
                        }
                        else if(pin >= 80 && pin <= 95){
                                addrM = addrMaster6;
                        }
                        else if(pin >= 96 && pin <= 111){
                                addrM = addrMaster7;
                        }
                        else if(pin >= 112 && pin <= 127){
                                addrM = addrMaster8;
                        }
```

```
                                else{
                                        addrM = addrMaster1;
                                }

                                if(pwm == 0){
                                        LED_ON;
                                        setPWM(addrM, pin, 4095, 0);
                                }
                                else{
                                        setPWM(addrM, pin, 0, pwm);
                                }
                                pin++;
                        }
                        uart_print(" Fin\r\n");
                }
        }
}


/************************************************************
** ARCHIVO: i2c.c                                          **
**                                                         **
** Libreria para la comunicación i2c del microprocesador   **
** ATMEGA32U4.                                             **
**                                                         **
** AUTOR: Jose Antonio Torre Albarsanz                     **
** FECHA: 20/05/2018                                       **
**                                                         **
** FUNCIONES:                                              **
**      - init_i2c()                                       **
**      - start_i2c()                                      **
**      - send_address_i2c(ADDR, w)                        **
**      - stop_i2c()                                       **
**      - send_i2c(data)                                   **
**      - read_i2c()                                       **
**      - readAck_i2c()                                    **
************************************************************/

#include <util/twi.h>

#include "i2c.h"
#include "PCA9685.h"

#ifndef CONFIG_DEFINES_H_
#define CONFIG_DEFINES_H_

#define SCL_CLOCK 400000L

#ifndef F_CPU
#define F_CPU 16000000UL //CPU Clock 16Mhz
#endif

#endif /* CONFIG_DEFINES_H_ */


/************************************************************
/                                                          **
/       Función para iniciar el bus I2C, asignando el puerto **
/       D0 para SCL y D1 para SDA. Asigna también la       **
/       frecuencia, y habilita el bus.                     **
/                                                          **
************************************************************/
void init_i2c() {
        PORTD &= ~(1 << 0); //Port D0 SCL
        PORTD &= ~(1 << 1); //Port D1 SDA
        TWBR = ((( F_CPU / SCL_CLOCK ) - 16) / 2);
        TWSR = 0;
        TWCR = ( 1 << TWEN ); // enable the i2c bus f
}
```

```
/*********************************************************
/                                                       **
/       Función para enviar la condición de inicio de     **
/       comunicación.                                     **
/                                                         **
*********************************************************/
void start_i2c() {
        //uint8_t   twstatus;
        // Envia la condición de inicio
        TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
        //Espera hasta que TWINT se ponga a 1. Esto indica que la condicion de inicio ha sido
          transmitida.
        while ( !(TWCR & (1<<TWINT)));

        //SIRVE PARA COMPROBAR SI HAY UN ERROR EN EL ENVIO DE LA CONDICION DE INICIO.
        /*twstatus = TW_STATUS & 0xF8;
        if ((TWSR & 0xF8) != TW_START){
                //ERROR();
                //print("\n   |Error: Start -> ");
                //phex(TWSR & 0xF8);
                //#endif
        }*/
}

/*********************************************************
/                                                         **
/       Función para enviar la dirección del esclavo y si   **
/       se va a escribir o leer.                          **
/                                                         **
/       PARAMETROS:                                       **
/       - ADDR -> Dirección del esclavo con el que nos vamos **
/                       comunicar.                         **
/       - w    -> Bit de escritura (0) o lectura (1).      **
/                                                         **
*********************************************************/
void send_address_i2c(uint8_t ADDR, uint8_t w) {
        //uint8_t   twstatus;
        TWDR = (ADDR<<1) | w ; //Carga la dirección mas el bit de escritura o lectura.
        TWCR = (1 << TWINT) | (1<<TWEN); //pone a 0 el bit TWINT para comenzar la transmisión de la
                                           dirección.

        while(!(TWCR & (1 << TWINT))); //Espera a que TWINT = 1
        //SIRVE PARA COMPROBAR SI HAY UN ERROR EN EL ENVIO DE LA DIRECCIÓN.
        /*twstatus = TW_STATUS & 0xF8;
        if ( (twstatus != TW_MT_SLA_ACK) && (twstatus != TW_MR_SLA_ACK) ) {
                //ERROR.
        }*/
}

/*********************************************************
/                                                       **
/       Función para enviar la condición de parada de la  **
/       comunicación.                                     **
/                                                       **
*********************************************************/
void stop_i2c() {
        TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
}

/*********************************************************
/                                                       **
/       Función para enviar el dato a la dirección enviada**
/       antes con la función send_address_i2c           **
/                                                       **
/       PARAMETROS:                                     **
/       - data -> Dato que vamos a enviar al esclavo    **
/                                                       **
*********************************************************/
void send_i2c(uint8_t data) {
        //uint8_t   twstatus;
        // send data to the previously addressed device
        TWDR = data; //Carga el dato en el registro TWDR.
        TWCR = (1 << TWINT) | (1<<TWEN); //Limpia TWINT en TWCR para comenzar la transmisión del
                                           dato.
```

```
        // wait until transmission completed
        while(!(TWCR & (1 << TWINT)));

        //SIRVE PARA COMPROBAR SI HAY UN ERROR EN EL ENVIO DEL DATO.
        /*twstatus = TW_STATUS & 0xF8;
        if( twstatus != TW_MT_DATA_ACK){
                //ERROR;
        }*/
}

/************************************************************
/                                                           **
/       Función para leer el dato recibido de la dirección **
/       enviada antes con la función send_address_i2c.      **
/                                                            **
************************************************************/
uint8_t read_i2c() {
        TWCR = (1 << TWINT) | (1 << TWEN);
        while(!(TWCR & (1 << TWINT)));
        return TWDR;
}

/************************************************************
/                                                           **
/       Función para leer el ACK                            **
/                                                           **
************************************************************/
int8_t readAck_i2c() {
        TWCR = (1 << TWINT) | (1 << TWEN) | (1<<TWEA);
        while(!(TWCR & (1 << TWINT)));
        return TWDR;
}

/************************************************************
** ARCHIVO: i2c.h                                          **
**                                                         **
** Libreria para la comunicación i2c del microprocesador  **
** ATMEGA32U4.                                             **
**                                                         **
** AUTOR: Jose Antonio Torre Albarsanz                     **
** FECHA: 20/05/2018                                       **
**                                                         **
** FUNCIONES:                                              **
**      - init_i2c()                                       **
**      - start_i2c()                                      **
**      - send_address_i2c(ADDR, w)                        **
**      - stop_i2c()                                       **
**  - send_i2c(data)                                       **
**  - read_i2c()                                           **
**  - readAck_i2c()                                        **
************************************************************/

#ifndef I2C_H_
#define I2C_H_

#include <util/twi.h>
#define I2C_READ 1
#define I2C_WRITE 0

void init_i2c();
void send_i2c(uint8_t);
uint8_t read_i2c();
int8_t readAck_i2c();
void send_address_i2c(uint8_t adr, uint8_t w);
void start_i2c();
void stop_i2c();

#endif /* I2C_H_ */
```

```
/*************************************************************
** ARCHIVO: PCA9685.c                                       **
**                                                          **
** Librería para el control del driver i2c PCA9685.        **
**                                                          **
** AUTOR: Jose Antonio Torre Albarsanz                      **
** FECHA: 20/05/2018                                        **
**                                                          **
** FUNCIONES:                                               **
**      - begin(_i2caddr)                                   **
**      - reset(_i2caddr)                                   **
**      - setPWMFre1(_i2caddr, freq)                        **
**      - setPWM(_i2caddr, num, on, off)                    **
**      - write_byte(_i2caddr, addr, d)                     **
**      - read_byte(_i2caddr, addr)                         **
*************************************************************/

#include "PCA9685.h"
#include "i2c.h"
#include <util/delay.h>
#include <stdint.h>


/*************************************************************
/                                                           **
/       Función para iniciar el driver PCA9685, en la cual **
/       primero inicia el i2c del micro y se resetea y      **
/       establece la frecuencia en la que va a funcionar el**
/       driver.                                             **
/                                                           **
*************************************************************/
void begin(uint8_t _i2caddr){
        init_i2c();
        reset(_i2caddr);
        // set a default frequency
        setPWMFreq(_i2caddr, 200);
}

/*************************************************************
/                                                           **
/       Función para resetear el driver.                    **
/       Se escribe en el registro MODE1 el valor 0x80       **
/                                                           **
*************************************************************/
void reset(uint8_t _i2caddr) {
        write_byte(_i2caddr, PCA9685_MODE1, 0x80);
        _delay_ms(10);
}

/*************************************************************
/                                                           **
/       Función para establecer la frecuencia de            **
/       funcionamiento del driver mediante el registro      **
/       PRESCALE.                                           **
/                                                           **
*************************************************************/
void setPWMFreq(uint8_t _i2caddr, float freq) {

        freq *= 0.9;  // Correct for overshoot in the frequency setting (see issue #11).
        float prescaleval = 25000000;
        prescaleval /= 4096;
        prescaleval /= freq;
        prescaleval -= 1;

        uint8_t prescale = floor(prescaleval + 0.5);

        uint8_t oldmode = read_byte(_i2caddr, PCA9685_MODE1);
        uint8_t newmode = (oldmode & 0x7F) | 0x10; // sleep
        write_byte(_i2caddr, PCA9685_MODE1, newmode); // go to sleep
        write_byte(_i2caddr, PCA9685_PRESCALE, prescale); // set the prescaler
        write_byte(_i2caddr, PCA9685_MODE1, oldmode);
        _delay_ms(5);
        write_byte(_i2caddr, PCA9685_MODE1, oldmode | 0xa0);  //This sets the MODE1 register to
                                                                turn on auto increment.
```

```
}

/*************************************************************
/                                                            **
/       Función para establecer el valor de pwm de una salida **
/       del driver, donde on es el momento de subida del     **
/       flanco, y off el momento de bajada del flanco.       **
/                                                            **
*************************************************************/
void setPWM(uint8_t _i2caddr, uint8_t pin, uint16_t on, uint16_t off) {

        start_i2c();
        send_address_i2c(_i2caddr, 0);
        send_i2c(LED0_ON_L+4*pin);
        send_i2c(on);
        send_i2c(on>>8);
        send_i2c(off);
        send_i2c(off>>8);
        stop_i2c();
}

/*************************************************************
/                                                            **
/       Función para escribir un byte d en una direccion     **
/       del driver addr.                                     **
/                                                            **
*************************************************************/
void write_byte(uint8_t _i2caddr, uint8_t addr, uint8_t d) {
        start_i2c();
        send_address_i2c(_i2caddr, 0);
        send_i2c(addr);
        send_i2c(d);
        stop_i2c();
}

/*************************************************************
/                                                            **
/       Función para leer un byte en una direccion           **
/       del driver addr.                                     **
/                                                            **
*************************************************************/
uint8_t read_byte(uint8_t _i2caddr, uint8_t addr) {
        start_i2c();
        send_address_i2c(_i2caddr, 0);
        send_i2c(addr);
        return read_i2c();
}

/*************************************************************
** ARCHIVO: PCA9685.h                                       **
**                                                          **
** Libreria para el control del driver i2c PCA9685.        **
**                                                          **
** AUTOR: Jose Antonio Torre Albarsanz                      **
** FECHA: 20/05/2018                                        **
**                                                          **
** FUNCIONES:                                               **
**      - begin(_i2caddr)                                   **
**      - reset(_i2caddr)                                   **
**      - setPWMFre1(_i2caddr, freq)                        **
**      - setPWM(_i2caddr, num, on, off)                    **
**      - write_byte(_i2caddr, addr, d)                     **
**      - read_byte(_i2caddr, addr)                         **
*************************************************************/


#ifndef PCA9685_H_
#define PCA9685_H_
#include <stdint.h>

#define PCA9685_SUBADR1 0x2
#define PCA9685_SUBADR2 0x3
#define PCA9685_SUBADR3 0x4
```

```
#define PCA9685_MODE1 0x0
#define PCA9685_PRESCALE 0xFE

#define LED0_ON_L 0x6
#define LED0_ON_H 0x7
#define LED0_OFF_L 0x8
#define LED0_OFF_H 0x9

#define ALLLED_ON_L 0xFA
#define ALLLED_ON_H 0xFB
#define ALLLED_OFF_L 0xFC
#define ALLLED_OFF_H 0xFD


void begin(uint8_t _i2caddr);
void reset(uint8_t _i2caddr);
void setPWMFreq(uint8_t _i2caddr, float freq);
void setPWM(uint8_t _i2caddr, uint8_t num, uint16_t on, uint16_t off);

uint8_t read_byte(uint8_t _i2caddr, uint8_t addr);
void write_byte(uint8_t _i2caddr, uint8_t addr, uint8_t d);


#endif /* PCA9685_H_ */

/************************************************************
** ARCHIVO: Xbee.c                                         **
**                                                         **
** Libreria para el uso del bus uart del microprocesador. **
**                                                         **
** AUTOR: Jose Antonio Torre Albarsanz                     **
** FECHA: 20/05/2018                                       **
**                                                         **
** FUNCIONES:                                              **
**      - uart_init(baud)                                  **
**      - uart_putchar(c)                                  **
**      - uart_getchar()                                   **
**      - uart_available()                                 **
**      - uart_print_P(*str)                               **
************************************************************/

#include "Xbee.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <stdint.h>


/************************************************************
/                                                          **
/       Función para iniciar el bus uart a la velocidad    **
/       indicada por el parametro de entrada baud, en baudios **
/                                                          **
************************************************************/
void uart_init(uint32_t baud)
{
        cli();
        UBRR1 = (F_CPU / 4 / baud - 1) / 2;
        UCSR1A = (1<<U2X1);
        UCSR1B = (1<<RXEN1) | (1<<TXEN1) | (1<<RXCIE1);
        UCSR1C = (1<<UCSZ11) | (1<<UCSZ10);
        tx_buffer_head = tx_buffer_tail = 0;
        rx_buffer_head = rx_buffer_tail = 0;
        sei();
}

/************************************************************
/                                                          **
/       Función para escribir un byte en el bus uart.      **
/                                                          **
************************************************************/
void uart_putchar(uint8_t c)
{
```

```
        uint8_t i;

        i = tx_buffer_head + 1;
        if (i >= TX_BUFFER_SIZE) i = 0;
        while (tx_buffer_tail == i) ; //Espera hasta que haya espacio en el buffer.
        //cli();
        tx_buffer[i] = c;
        tx_buffer_head = i;
        UCSR1B = (1<<RXEN1) | (1<<TXEN1) | (1<<RXCIE1) | (1<<UDRIE1);
        //sei();
}

/*********************************************************
/                                                       **
/       Función para leer un byte del buffer            **
/                                                       **
*********************************************************/
uint8_t uart_getchar(void)
{
        uint8_t c, i;

        while (rx_buffer_head == rx_buffer_tail) ; // wait for character
        i = rx_buffer_tail + 1;
        if (i >= RX_BUFFER_SIZE) i = 0;
        c = rx_buffer[i];
        rx_buffer_tail = i;
        return c;
}

/*********************************************************
/                                                        **
/       Función que devuelve el numero de bytes disponibles  **
/       en el buffer. Llamar a esta función antes de leer un **
/       byte para esperar a que haya un byte en el buffer.   **
/                                                         **
*********************************************************/
uint8_t uart_available(void)
{
        uint8_t head, tail;

        head = rx_buffer_head;
        tail = rx_buffer_tail;
        if (head >= tail) return head - tail;
        return RX_BUFFER_SIZE + head - tail;
}
/*********************************************************
/                                                       **
/       Interrupcion para Transmitir                     **
/                                                        **
*********************************************************/
ISR(USART1_UDRE_vect)
{
        uint8_t i;

        if (tx_buffer_head == tx_buffer_tail) {
                UCSR1B = (1<<RXEN1) | (1<<TXEN1) | (1<<RXCIE1);
        }
        else {
                i = tx_buffer_tail + 1;
                if (i >= TX_BUFFER_SIZE) i = 0;
                UDR1 = tx_buffer[i];
                tx_buffer_tail = i;
        }
}

/*********************************************************
/                                                       **
/       Interrupcion para Recibir                        **
/                                                        **
*********************************************************/
ISR(USART1_RX_vect)
{
        uint8_t c, i;
```

```
        c = UDR1;
        i = rx_buffer_head + 1;
        if (i >= RX_BUFFER_SIZE) i = 0;
        if (i != rx_buffer_tail) {
                rx_buffer[i] = c;
                rx_buffer_head = i;
        }
}


#define BAUD_RATE 9600

#define CPU_PRESCALE(n) (CLKPR = 0x80, CLKPR = (n))

// write a string to the uart
#define uart_print(s) uart_print_P(PSTR(s))

/************************************************************
/                                                           **
/       Función para escribir una cadena de caracteres en el **
/       buffer.                                             **
/                                                           **
************************************************************/
void uart_print_P(const char *str)
{
        char c;
        while (1) {
                c = pgm_read_byte(str++);
                if (!c) break;
                uart_putchar(c);
        }
}

/************************************************************
** ARCHIVO: Xbee.c                                         **
**                                                         **
** Libreria para el uso del bus uart del microprocesador. **
**                                                         **
** AUTOR: Jose Antonio Torre Albarsanz                     **
** FECHA: 20/05/2018                                       **
**                                                         **
** FUNCIONES:                                              **
**      - uart_init(baud)                                  **
**      - uart_putchar(c)                                  **
**      - uart_getchar()                                   **
**      - uart_available()                                 **
**      - uart_print_P(*str)                               **
************************************************************/

#ifndef XBEE_H_
#define XBEE_H_

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <stdint.h>


// These buffers may be any size from 2 to 256 bytes.
#define RX_BUFFER_SIZE 64
#define TX_BUFFER_SIZE 40
#define F_CPU 16000000

#define LED_CONFIG      (DDRD |= (1<<6))
#define LED_OFF          (PORTD &= ~(1<<6))
#define LED_ON         (PORTD |= (1<<6))
#define BAUD_RATE 9600

#define CPU_PRESCALE(n) (CLKPR = 0x80, CLKPR = (n))

// write a string to the uart
#define uart_print(s) uart_print_P(PSTR(s))

static volatile uint8_t tx_buffer[TX_BUFFER_SIZE];
```

```
static volatile uint8_t tx_buffer_head;
static volatile uint8_t tx_buffer_tail;
static volatile uint8_t rx_buffer[RX_BUFFER_SIZE];
static volatile uint8_t rx_buffer_head;
static volatile uint8_t rx_buffer_tail;


void uart_init(uint32_t baud);
void uart_putchar(uint8_t c);
uint8_t uart_getchar(void);
uint8_t uart_available(void);
void uart_print_P(const char *str);

#endif /* XBEE_H_ */
```

PRESUPUESTO

EJECUCIÓN MATERIAL	
Compra de ordenador personal (S.O. incluido)	1.000€
Licencia de *Altium Designer*	1.200€
Material de oficinal	150€
Teensy 2.0	16€
Kit inicio Xbee S1 Pro	60€
8 x PCA9685	8€
Componentes electrónicos (resistencias, condensadores, etc)	30€
Fabricación PCBs	320€
GASTOS GENERALES	
16% sobre Ejecución Material	394€
BENEFICIO INDUSTRIAL	
6% sobre Ejecución Material	148€
HONORARIOS PROYECTO	
750 horas a 30€/hora	22.500€
MATERIAL FUNGIBLE	
Gastos de impresión	100€
Encuadernación	50€
SUBTOTAL PRESUPUESTO	**25.976€**
21% Subtotal Presupuesto	5.454,96€
TOTAL PRESUPUESTO	**31.430,96€**

Madrid, Septiembre de 2018

El Ingeniero Jefe de Proyecto

Fdo.: Jose Antonio Torre Albarsanz.
Ingeniero de Telecomunicación

PLIEGO DE CONDICIONES

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un SISTEMA DE SUSTITUCIÓN SENSORIAL. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el

deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad "Presupuesto de Ejecución de Contrata" y anteriormente llamado "Presupuesto de Ejecución Material" que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.